

**MOTION-PLANNING AND CONTROL OF AUTONOMOUS VEHICLES
TO SATISFY LINEAR TEMPORAL LOGIC SPECIFICATIONS**

by
Zetian Zhang

A thesis submitted to the Faculty of

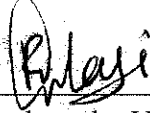
WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY IN AEROSPACE ENGINEERING.

November 2018

APPROVED:



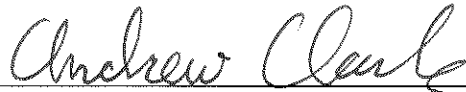
Dr. Raghvendra V. Cowlagi, Advisor
Assistant Professor, Aerospace Engineering Program.



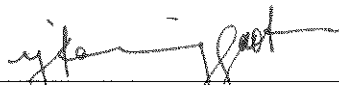
Dr. Michael A. Demetriou, Committee Member
Professor, Aerospace Engineering Program.



Dr. Jie Fu, Committee Member
Assistant Professor, Robotics Engineering Program.



Dr. Andrew Clark, Committee Member,
Assistant Professor, Electrical and Computer Engineering Program.



Dr. Nikhil Karanjgaokar, Graduate Committee Representative,
Assistant Professor, Aerospace Engineering Program

*To my parents and grandparents
for their unconditional love*

Acknowledgements

There are many people that have earned my gratitude for their contribution to my time as a Ph.D. student. More specifically, I would like to thank three groups of people, without whom this thesis would not have been possible: my dissertation committee members, my lab mates, and my family and friends.

My Advisor

First and foremost I want to thank my advisor Raghvendra V. Cowlagi. It has been an honor to be his first Ph.D. student. It all started in Fall 2013 when he offered me such a great opportunity to make my life a difference. On the academic level, Raghu taught me fundamentals of conducting scientific research in the control area and gave me research directions. Under his supervision, I learned how to define a research problem, find a solution to it, and finally publish the results. On a personal level, Raghu inspired me by his hardworking and passionate attitude and he always believed in me like nobody else. To summarize, I would give Raghu most of the credit for all his contributions of time, ideas, and funding to make my Ph.D. experience productive and stimulating.

Dissertation Committee Members

Besides my advisor, I would like to thank the rest of my dissertation committee members (Michael A. Demetriou, Jie Fu, and Andrew Clark) for their great support and invaluable advice. I am thankful to Prof. Demetriou for his crucial remarks that shaped my final dissertation. Thanks Prof. Fu for her inspiration and valuable discussions. Thanks Prof. Clark for agreeing to serve on my dissertation committee on such a short notice and his insightful comments.

My Labmates

I would like to thank my lab mates for their continued support. This dissertation would not have been possible without the intellectual contribution of Ruixiang Du, Jie Fang and Benjamin Cooper. They also have been great friends, so many valuable discussions and comments topics ranging from life to academic. Thanks Haocheng Li for all his passionate math lectures. Thanks Jighjigh Tersoo-Ivase for many fun during my first year.

My Family and Friends

I owe a lot to my parents, who encouraged and helped me at every stage of my life, and longed to see this achievement come true. I deeply miss my grandparents, who are not with me to share this joy. Thanks my brother Lin and sister Boya for their support. Special thanks to Yang Song and all my friends outside WPI for accompanying me in this special period of my life.

Abstract

Motion-planning is an essential component of autonomous aerial and terrestrial vehicles. The canonical motion-planning problem, which is widely studied in the literature, is of planning point-to-point motion while avoiding obstacles. However, the desired degree of vehicular autonomy has steadily risen, and has consequently led to motion-planning problems where a vehicle is required to accomplish a high-level intelligent task, rather than simply move between two points. One way of specifying such intelligent tasks is via linear temporal logic (LTL) formulae. LTL is a formal logic system that includes temporal operators such as *always*, *eventually*, and *until* besides the usual logical operators. For autonomous vehicles, LTL formulae can concisely express tasks such as persistent surveillance, safety requirements, and temporal orders of visits to multiple locations.

Recent control theoretic literature has discussed the generation of reference trajectories and/or the synthesis of feedback control laws to enable a vehicle to move in manners that satisfy LTL specifications. A crucial step in such synthesis is the generation of a so-called discrete abstraction of a vehicle kinematic / dynamic model. Typical techniques of generating a discrete abstraction require strong assumptions on controllability and/or linearity. This dissertation discusses fast motion-planning and control techniques to satisfy LTL specifications for vehicle models with nonholonomic kinematic constraints, which do not satisfy the aforesaid assumptions. The main contributions of this dissertation are as follows.

First, we present a new technique for constructing discrete abstractions of a Dubins vehicle model (namely, a vehicle that moves forward at a constant speed with a minimum turning radius). This technique relies on the so-called method of lifted graphs and precomputed reachable set calculations. Using this technique, we provide an algorithm to generate vehicle reference trajectories satisfying LTL specifications without requiring complete controllability in the presence of workspace constraints, and without requiring linearity or linearization

of the vehicle model.

Second, we present a technique for centralized motion-planning for a team of vehicles to collaboratively satisfy a common LTL specification. This technique is also based on the method of lifted graphs.

Third, we present an incremental version of the proposed motion-planning techniques, which has an "anytime" property. This property means that a feasible solution is computed quickly, and the iterative updates are made to this solution with a guarantee of convergence to an *optimal* solution. This version is suited for real-time implementation, where a hard bound on the computation time is imposed.

Finally, we present a randomized sampling-based technique for generating reference trajectories that satisfy given LTL specifications. This technique is an alternative to the aforesaid technique based on lifted graphs.

We illustrate the proposed techniques using numerical simulation examples. We demonstrate the superiority of the proposed techniques in comparison to the existing literature in terms of computational time and memory requirements.

Contents

1	Introduction	1
1.1	Literature Review	2
1.1.1	High-level Tasks in Motion-planning	2
1.1.2	Low-level Trajectory Generation in Motion-planning	3
1.1.3	Motion-planning for Satisfying LTL Specifications	5
1.2	Thesis Overview and Statement of Contributions	8
1.2.1	Overview	8
1.2.2	Contributions	10
2	Motion-planning for a Nonholonomic Vehicle	13
2.1	Problem Formulation	13
2.1.1	Vehicle Model	13
2.1.2	Workspace partition	14
2.1.3	LTL _{-X} specifications	15
2.2	Lifted Graph	17
2.2.1	Edge Transition Costs in \mathcal{G}_H	19
2.3	Product Transition System	22
2.3.1	Route-Planning Algorithm	25
2.4	Numerical Computation of Edge Transition Costs	27
2.5	Illustrative Numerical Simulation Results	34
2.5.1	Discussion	39
3	Motion-planning for Multiple Nonholonomic Vehicles	43
3.1	Problem Formulation	43
3.1.1	Regions of Interest Graph	44
3.1.2	Global LTL _{-X} specifications	45
3.2	Multi-Vehicle Motion-planning	46

3.3	Illustrative Numerical Simulation Results	49
4	Incremental Motion-Planning	55
4.1	Illustrative Examples and Discussion	60
4.1.1	o	62
5	Sampling Based Motion-Planning	66
5.1	Problem Formulation	66
5.2	LTL Satisfaction by State Trajectories	67
5.3	Proposed Solution to Problem 3	69
5.3.1	Description of Subroutines	71
5.3.2	Sampling Heuristic	74
5.4	Numerical Simulation Examples	75
5.4.1	Example Neglecting Kinematical Constraints	75
5.4.2	Examples with Fixed-Wing Aircraft Model	77
5.4.3	Example with Quadrotor Aircraft Dynamical Model	81
5.5	Extension to Multiple Vehicles	83
6	Conclusions and Directions of Future Work	85

List of Figures

1.1	Illustration of cell decomposition and the associated graph.	4
2.1	Example of a lifted graph (the vertices $(3, 2)$ and $(5, 2)$ of the lifted graph are drawn twice for clarity of the diagram).	18
2.2	Conceptual illustration of the sets \mathcal{R} , $\mathcal{Q}(\cdot)$ and $\mathcal{S}(\cdot)$ for $H = 2$. The large arrows indicate the existence of admissible state trajectories as described in the text preceding (2.6).	20
2.3	Conceptual illustration of the proposed route-planning approach.	23
2.4	Pseudo-code for executing a modified form of Dijkstra’s algorithm on the product automaton.	27
2.5	Illustration of canonical traversals (adjacent and opposite) and local coordinate axes systems. In (a) and (b), the green- and red colored dotted lines indicate, respectively, the entering and exiting faces of traversal of the cell.	28
2.6	Tile library for $H = 2$, auto-generated using a MATLAB [®] script.	30
2.7	Tile library for $H = 3$, auto-generated using a MATLAB [®] script.	30
2.8	Application of the proposed approach: illustration of the effects on the resultant path of changes in the control input constraint. The numerical values ρ are in dimensionless distance units, where 1 unit is equal to the side of a cell in the uniform decomposition.	35
2.9	Application of the proposed approach: illustration of the effects on the resultant path of changes in the initial state. Here, the LTL _{-X} specification is ϕ_3 (described in text). The numerical values of position coordinates and of ρ are in dimensionless distance units, where 1 unit is equal to the side of a cell in the uniform decomposition.	36

2.10	Additional examples that illustrate the significant impact of nonholonomic motion constraints on the manner in which the given LTL specifications are satisfied (details in text).	38
3.1	Conceptual relationship between the various graphs involved.	45
3.2	An instance of Problem 1. Here, ROIs λ_1 and λ_2 are indicated in red (cells 19 and 27, respectively), and ROIs λ_3 and λ_4 are indicated in gray (cells 10–12 and 16–18, respectively).	46
3.3	Pseudocode description of the proposed incremental algorithm for solving the multi robot task/path planning problem.	49
3.4	Solutions to the motivating example in Fig. 3.2. The yellow- and blue-colored cells indicate, respectively, the paths planned for the first and second vehicle (additional details are in Section 5.4).	52
3.5	Illustration of motion-planning for 4 vehicles. The LTL specification is similar to that of the example in Fig. 3.2. Here, ROIs to be always avoided are indicated in gray, and ROIs to be eventually visited are indicated in red (order of visit is not relevant). The base locations are indicated in green, and the paths found for each vehicle are indicated in yellow.	53
3.6	Illustration of motion-planning when temporal synchronization is important. The paths of the two vehicle are indicated with yellow- and blue-colored cells.	54
4.1	Illustration of proposed implementation of the H-cost framework with incremental planning	56
4.2	Pseudocode description of the proposed incremental algorithm for solving the H -cost optimal path problem.	58
4.3	Illustrative example of solution of the H -cost optimal path problem using incremental path repair.	61
4.4	Illustration of replanning in response to changes in the environment.	64

4.5	Illustrative example: during intermediate iterations, feasible solutions are available, whereas the solution cost is reduced.	65
5.1	Pseudocode description of the proposed sampling based algorithm for solving the task/path planning problem.	70
5.2	Pseudocode description of the Update random tree subroutine.	73
5.3	Pseudocode description of sampling heuristic subroutine.	74
5.4	Workspace (a) and product space (b) for the example in Section 5.4.1. . . .	76
5.5	Performance of sampling heuristic for the example in Section 5.4.1.	77
5.6	Results for Example 1.	78
5.7	Simulation results for specification ϕ_1 in Example 1 with the fixed wing aircraft model (5.1).	79
5.8	Simulation results for specification ϕ_2 in Example 2 with the fixed wing aircraft model (5.1).	80
5.9	Simulation results for specification ϕ_3 illustrating trajectories conditional on properties of the environment.	81
5.10	Simulation results for specification ϕ_3 illustrating quadrotor trajectories conditional on properties of the environment.	82
5.11	Performance of sampling heuristic for specification ϕ_3	82
5.12	Simulation result for specification $\phi_1 = \diamond\lambda_1 \wedge \diamond\lambda_2 \wedge \diamond\lambda_3$	84

List of Tables

2.1	Number of vertices in the lifted graph, for various values of the parameter H , and the time required to record the graph topology.	39
2.2	Execution times for the numerical simulation examples discussed in Section 5.4. The blank entries for τ_{search}^0 and τ_{search}^1 for $H = 3, 4$ indicate that the search returned failure after a finite number of iterations, and no route was found. .	40

Chapter 1

Introduction

The demand for higher degrees of autonomy in unmanned aerial and terrestrial vehicles (UXVs) widens the scope of onboard guidance from the traditional task of optimal waypoint navigation to higher-level tasks involving intelligent decision-making. We discuss the role of linear temporal logic (LTL) specifications in formulating such higher-level tasks, and report a new guidance algorithm capable of satisfying LTL specifications on aircraft motion.

LTL is a formal system, similar to the commonly used propositional logic system. In addition to the standard operators `and`, `or`, and `not`, LTL includes temporal operators such as `always`, `eventually`, and `until`. In contrast to propositional logic, LTL can be used to express properties of an infinite series of computations, which is useful for expressing vehicular tasks such as persistent surveillance. LTL has been used in software design for the specification of “correct” behaviors of algorithms [1, 2]. Software algorithms can be examined to determine if they meet given LTL specifications, using so-called *formal verification* methods such as model checking [3].

More recently, LTL has been applied for specifying behaviors of dynamical systems, and in particular, behaviors of mobile vehicles [4–6]. For robotic vehicles such as UXVs, high-level intelligent “tasks” as well as properties of safe behaviors, e.g. “*perform persistent surveillance in region A until a target is found, then report data to region B, never fly in region C, and finally return to base*” can be formulated with LTL specifications. Trajectory generation and tracking algorithms are then required to plan and execute UAV motions to satisfy these specifications. The state-of-the-art approaches to control subject to LTL specifications do not suffice to address the needs of aircraft kinematic and dynamic constraints.

1.1 Literature Review

Motion-planning and control for UXVs is extensively studied in the literature. In what follows, we use the term *trajectory* to refer to a locus of points in the state space of the vehicle model, and the term *route* or the term *path* to refer to a discrete representation of a trajectory (e.g., a sequence of waypoints or regions). The term *route-planning* (synonymous with *path-planning*) is the process of finding a sequence of regions in the workspace that enclose the vehicle’s desired trajectory. The term *motion-planning* refers to route-planning to satisfy a higher-level task requirement, along with the generation of a lower level reference trajectory to execute the planned route. The following are highlights of the various topics studied in this literature.

1.1.1 High-level Tasks in Motion-planning

Vehicle routing algorithms address the high-level task of optimally touring a set of target locations [7, 8], i.e. generalized versions of the traveling salesperson problem. These problems are formulated as combinatorial optimization problems [9] (e.g. optimal tour on a graph). In stochastic-dynamic VRPs, a *policy* is determined, rather than a single tour [10, 11]. Recently, the incorporation of vehicle kinematic characteristics have been investigated for vehicle routing problems [12, 13].

Coverage and coordination algorithms address the high-level task of deploying multiple autonomous vehicles over a region for the purposes of, say, surveillance or information gathering [14–16]. These problems are analyzed with tools from computational geometry (e.g. Voronoi diagrams for coverage control) and graph theory (e.g. for analyzing vehicle communication network topologies). The vehicle models considered are typically simple kinematical models.

We focus on high-level tasks described by LTL specifications, which we discuss in greater detail in Section 1.1.3.

The **point-to-point motion-planning** problem refers to trajectory generation where the high-level task is to move from a prespecified initial point to a prespecified destination point while avoiding obstacles in the environment. Hierarchical approaches are often used to separate this problem into a geometric route-planning problem, followed by a continuous trajectory optimization problem [17–24]. The route-planner is mainly concerned with obstacle avoidance, and is usually formulated as a discrete optimization problem of finding a path with lowest cost in a graph. The trajectory planner then attempts to find a control inputs for the vehicle to execute the geometric path found by the route planner.

Three broad ideas have emerged in **geometric path-planning** [25, 26]: roadmaps, and cell decompositions, and artificial potential fields. The proposed technical approach is based on cell decompositions [26, Ch. 5], [27]. These involve a partitioning of the environment into convex, non-overlapping 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 \mathcal{G} 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 pre-specified initial cell to a pre-specified goal cell in the graph \mathcal{G} then corresponds to a sequence of FREE cells from the initial cell to the goal cell in the graph \mathcal{G} (see Fig. 1.1). Triangular and trapezoidal decompositions [26, Ch.6], [28] are widely used exact cell decomposition techniques for environments with polygonal obstacles, whereas quadtree-based methods [29–31] (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 [32–34].

1.1.2 Low-level Trajectory Generation in Motion-planning

Variational optimal control theory has investigated trajectory optimization for aircraft and spacecraft with applications of the Minimum Principle [35–39]. The domain of optimization is typically a space of piecewise continuous functions of \mathbb{R}_+ . The optimization is

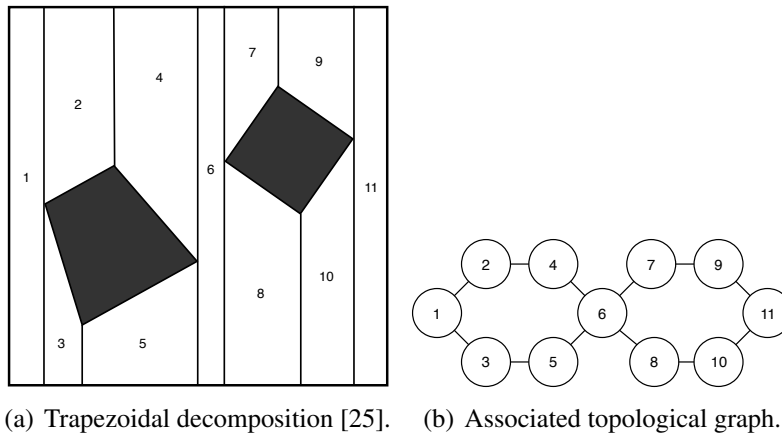


Figure 1.1: Illustration of cell decomposition and the associated graph.

subject to constraints, including those of an ordinary differential equation that describes the evolution of a dynamical system (e.g., aircraft motion).

Well-known examples in trajectory optimization include the problem of finding a shortest path for a vehicle with constant speed and a fixed minimum turn radius [40–43] (also known as the Dubins vehicle) and the so-called Zermelo navigation problem of finding a heading profile to minimize time of traversal for a vehicle moving in a drift field [35, 44, 45]. Trajectory optimization problems with state inequality constraints typically do not admit closed-form analytical solutions, and numerical methods are employed [46, 47]; application examples can be found, for instance in [44, 48–51]. Optimal control theory has also been applied to find minimum-time speed profiles on pre-specified geometric paths [52–55] with input- and vehicle dynamical constraints. Feedback optimal control techniques other than LQR [56] are typically variants of dynamic programming [57, 58].

References [52–55] discuss time-optimal trajectory planning along pre-specified geometric paths for specific vehicle dynamics. Other related works in the literature include [59], which uses a special history-based cost; [60], which deals with kinodynamic planning for robotic manipulators; [61], which uses a hybrid model to describe the motion of a rotorcraft in terms of preprogrammed maneuvers; and [62], which discusses trajectory planning based on the solution of the Hamilton-Jacobi-Bellman equation.

Path-planning for nonholonomic vehicle models dates back to the seminal work of Dubins [63], who proved that minimum-length curves of bounded curvature that connect two prespecified points in the plane with prespecified initial and final tangent directions must belong to a finite family of curves. This family of curves is characterized by concatenations of up to three circular arc and straight line segments, where the radius of the circular arcs is equal to the inverse of the given curvature bound. This result has been applied for path-planning for nonholonomic vehicles that can move forwards only with a bounded rate of turn (e.g. fixed-wing aircraft flying at a constant altitude and speed). Reeds and Shepp [64] extended this result further for nonholonomic vehicles that can move both forwards and backwards (often called *car-like* vehicles). It is difficult to find shortest paths with continuous curvature even without the obstacle-avoidance requirement and is proven to be NP-hard [65].

Randomized sampling-based methods for path-planning and trajectory generation have been studied in the last two decades. Probabilistic roadmap (PRM) methods [66–70], [26, Ch. 7], and methods that use rapidly exploring trees (RRT) (RRTs) [71–76] can address the vehicle’s kinematic and dynamic constraints, but often result in suboptimal trajectories. 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. PRM is applied to nonholonomic car-like robots path planning in [77], and is shown to be probabilistically complete [78, 79].

1.1.3 Motion-planning for Satisfying LTL Specifications

Linear Temporal Logic (LTL) is used in computer science to specify desired characteristics of algorithms [1]. The application of temporal logic in robotics dates back to [80]. Recently LTL has been applied for specifying behaviors of mobile vehicles. In [81] the author designed controllers using navigation functions that satisfy LTL formulas. In [4], a framework has been presented for generating feedback-control laws for planar

robots operating in polygonal environments. In [5], the author designed closed-loop hybrid controllers that guarantee the generation of continuous robot trajectories that satisfy temporal specifications. In [82], author provide a fully automated framework for control of linear systems with specifications given in terms of LTL formulas. In [6], the author summarize their research trend to use symbolic techniques for robot motion planning and control. For the extension work of this topic, in [83], the author describe a framework for multi-robot motion planning using computation tree logic formulas, but without considering the dynamics of the robots. Centralized motion-planning algorithms for the team are developed in [84].

Several results on the control of dynamical systems to satisfy LTL specifications are reported in the literature. All of these results crucially rely on generating a *discrete abstraction* of a dynamical system [85, 86], which is a finite state transition system whose transition sequences are equivalent – via appropriately defined equivalence relations – to admissible state trajectories of the dynamical system. Discrete abstractions allow the application of formal verification and/or search algorithms to find transition sequences that satisfy the given LTL specifications. To this end, a compact domain of interest in the state space of the dynamical system Γ is partitioned into a finite number of regions. Each region in this partition is uniquely associated with a state of a transition system \mathcal{G} . Control laws are designed to steer trajectories of Γ between these regions and thereby emulate state transitions in \mathcal{G} [4, 81, 87]. The LTL specification is represented by a *Büchi automaton* \mathcal{B} , which is a finite state transition system with transition sequences exactly equal to those admissible under the given LTL specification [88–91]. Finally, a *product transition system* of \mathcal{B} and \mathcal{G} is searched. Any transition sequence of this product system can be projected to a path in \mathcal{G} , which in turn can be associated with control laws and admissible state trajectories of Γ . These state trajectories are therefore guaranteed to satisfy the given LTL specification.

The preceding approach is in general beneficial, and is considered “canonical” to the extent that many works in the literature assume a priori the existence of a finite transition system that represents an underlying dynamical system [84, 92, 93]. However, there are several serious shortcomings in the state-of-the-art, which the proposed dissertation seeks to

address.

First, the efficient construction of discrete abstractions is largely restricted to low-dimensional linear systems [87, 94, 95]. The approach of state-space partitioning naturally incurs the “curse of dimensionality”, and leads to an explosion in the number of states in \mathcal{G} . The situation is worse for nonlinear systems, where the requirement of finding local control laws to steer state trajectories between adjacent regions in the aforesaid partition leads to enormous numbers of states in \mathcal{G} , even for low-dimensional systems. Consider for example a recently reported work [96], where the discrete abstraction of a three-state vehicle model consists of 91,035 states and over 34 *million* transitions. Whereas the size of the discrete abstraction may not be of high relevance for offline verification purposes, it can easily overwhelm the limited computational resources onboard UAVs. Several other recent works address the satisfaction of LTL specifications for nonlinear systems, but either involve linearization [97], or rely on strong controllability properties or special vector field structure of Γ [98, 99] that are not applicable to the aircraft model considered in this paper.

Second, the state-of-the-art methods for satisfying LTL specifications do not “adapt” to changes in model parameters that affect motion characteristics, e.g. bounds on the steering rate. When such parameters do change, the entire process of computing the discrete abstraction and product transition system must be repeated, which can be computationally expensive or prohibitive.

Third, the state-of-the-art methods do not accommodate standard trajectory optimization algorithms, which have been established for aircraft guidance [46]. This lack of context to traditional aircraft guidance algorithms therefore restricts the scope of practical application of these methods.

The literature on **multi-vehicle teams subject to LTL specifications** falls into two broad categories: (1) global specifications for the entire team are assumed, and centralized motion-planning algorithms are developed [84, 94], and (2) separate specifications for individual vehicles in the team are assumed, and distributed motion-planning algorithms are

developed [100, 101]. Distributed algorithms for satisfying global specifications are yet an open subject of research. We seek to extend the results of [84] to include nonholonomic kinematic constraints. A crucial assumption in [84] is that a discrete abstraction of the vehicle dynamical model is readily available. This assumption is valid for holonomic vehicles that can be modeled as single or double integrators. However, for nonholonomic vehicles, the generation of a finite state model of the vehicle’s motion is non-trivial, and is addressed in this dissertation.

1.2 Thesis Overview and Statement of Contributions

The goal of this dissertation is to develop fast motion-planning and control algorithms that enable autonomous vehicles to satisfy LTL specifications.

1.2.1 Overview

In chapter 2, we propose a novel and computationally efficient approach to aircraft guidance subject to LTL specifications, where nonholonomic constraints on the vehicle’s motion are considered. The scope of applications of this work primarily involves small-scale UAVs in domains such as urban environments where the dimensions of the UAV maneuvering characteristics (e.g. minimum turn radius) are comparable to the dimensions of workspace features (e.g. distances between obstacles). The proposed approach is based on workspace partitioning, and relies on the idea of *lifted graphs* [102]. Briefly, edges in a lifted graph are successions of adjacent edges in the topological graph associated with the workspace partition. We associate edges of the lifted graph with reachability properties of the aircraft model. A finite state transition system is then constructed as the product of the lifted graph with the Büchi automaton associated with the given LTL specification. Each run of this product transition system has a unique projection on the collection of paths in the workspace partition graph. We show that each run of the product transition system is associated with

an admissible state trajectory of the aircraft model that satisfies the given LTL specifications and the proposed guidance algorithm relies on searching the product transition system.

In chapter 3, we propose a centralized motion-planning algorithm for a team of robotic vehicles subject to nonholonomic kinematic constraints and global LTL specifications. The problem formulation relies on workspace cell decompositions, where certain regions of interest in the robots' shared workspace are defined. The proposed algorithm involves two graphs: first, the topological graph \mathcal{G} arising from the workspace cell decomposition, and second, a graph \mathcal{G}^R arising from vertex aggregation on \mathcal{G} , such that each region of interest is a vertex \mathcal{G}^R . The main technical innovation in the proposed algorithm is the application of the *method of lifted graphs* which defined in chapter 2 to determine feasibility of edge transitions in \mathcal{G} and \mathcal{G}^R . As in [84], a team transition system is first established. Next, a product transition system is constructed from this team transition system and the Büchi automaton associated with the global LTL specifications. Runs of this product transition system can be uniquely projected to paths (for each vehicle) that are compatible with the nonholonomic constraints and also ensure that the global LTL specifications are satisfied.

In chapter 4, we propose the idea of *incremental H-cost* motion-planning. The proposed algorithm retains the primary benefit of the original *H-cost* technique [102] of finding an optimal high-level plan with the guarantee that there exists a state trajectory of Γ to execute this plan. Additionally, the proposed algorithm significantly mitigates the computational cost of the original *H-cost* approach by introducing incremental computations. The proposed algorithm is incremental in that it produces a sequence of *feasible* plans in intermediate iterations, which asymptotically converge to an *optimal* plan. Therefore, the proposed algorithm is *eventually optimal*, and it is also suitable for real-time implementation with a hard bound on the available computation time. The primary motivation behind developing an incremental algorithm is as follows: the fundamental computational complexity in the *H-cost* technique (or any similar hierarchical motion-planning technique) cannot be defeated if the objective is solely to find an *optimal* motion plan. In the interests of real-time implementation, it is beneficial to develop a motion planner that returns a feasible solution at

nearly any time during its search for an optimal motion plan. The proposed algorithm relies on iterative repair of paths in lifted graphs. We focus on the point-to-point motion-planning problem for developing the algorithm; as previously stated, it is easy to envision extensions to the general problem where the high-level task is specified by logic formulae. The H -cost algorithm [102] attempts to find optimal paths in lifted graphs with appropriately defined edge costs. Whereas the search for such optimal paths is inherently difficult, the proposed approach alleviates this difficulty by iteratively “seeding” the search with the result of a simpler optimization problem, and then replacing high-cost edges in the “seed” path. This idea of path repair has been discussed in different contexts in [103, 104].

In chapter 5, we propose a randomized sampling-based motion planning algorithm to satisfy LTL specifications. Similar to RRT*, the proposed algorithm incrementally generates a tree abstraction of the vehicle dynamical system model. Random samples are taken from multiple copies of the state space, with each copy uniquely associated with a state of the Büchi automaton. The proposed algorithm retains the properties of RRT*: namely, the proposed algorithm is probabilistically complete (i.e. guaranteed to find a trajectory satisfying the specifications if it exists) and asymptotically optimal. To achieve significant reductions in computation time, we propose a sampling heuristic that provides a bias for growing the tree structure. This sampling heuristic preserves the completeness and optimality of the algorithm. We provide numerical simulation results of the application of the proposed algorithm to a fixed-wing aircraft kinematical model and to a quadrotor aircraft dynamical model.

1.2.2 Contributions

We make the following contributions toward developing motion-planning algorithm to satisfy LTL specifications.

Graph based guidance algorithm to find trajectories satisfying LTL specifications:

The state-of-the-art in this area involves linearization or feedback linearization (in the absence of state- or input constraints), followed by discrete abstraction of the linearized model.

The proposed algorithm does not involve linearization. The proposed algorithm relies on partitioning the *output space* (instead of the state space), which reduces the number of states and transitions in the aforesaid product transition system. Previous attempts at using output space decompositions [4] have either ignored state- and control input constraints and/or rely on strong controllability properties of the dynamical system such as controllability in the presence of workspace constraints [98], which is not true of the aircraft model considered in our work. We also emphasize offline preprocessing of a significant portion of the computations in the proposed route-planning algorithm, thereby elevating the practical applicability of the proposed approach in future UAV onboard, real-time guidance systems. The proposed algorithm accommodates independent trajectory optimization algorithms for generating reference trajectories. Also, in contrast to the state-of-the-art [4,87,105], the proposed approach does not use up control authority for the sake of creating a discrete abstraction of the vehicle model. Furthermore, we discuss a local trajectory generation problem which can be solved by standard numerical trajectory optimization tools. These two features of the proposed approach not only afford the flexibility of using the proposed algorithm in conjunction with existing trajectory optimization algorithms for UAVs, but also allow the use of the full range of a UAV’s maneuvering capabilities.

Multiple vehicle motion-planning: We present an novel motion-planning method to enable a team of nonholonomic vehicles to satisfy global LTL specifications. This method relies on vertex aggregation in the team’s shared workspace, and on the new method of lifted graphs. We propose an incremental algorithm computing the desired motion plan and the proposed approach promises better scalability to higher-dimensional vehicle dynamical models.

Incremental motion-planning: We propose a novel, computationally efficient, and hierarchically separated solution to the point-to-point motion-planning problem. The proposed approach can incorporate complex vehicle dynamical characteristics, and it guarantees “compatibility” between the task-planning and trajectory generation algorithms. Furthermore, the proposed approach possesses the desirable property of maintaining feasible solutions in in-

intermediate iterations while asymptotically converging to an optimal solution. Secondly, we elucidate the properties of lifted graphs as a general tool for hierarchical motion-planning involving different task-planning problems and vehicle dynamics. Finally, we discuss the suitability of the proposed approach to future real-time computations with bounds on the computation time. Specifically, the proposed approach can be implemented to use as much computational time as available, instead of requiring a significant minimum period for arriving at a usable result.

Randomized sampling-based algorithm to find trajectories satisfying LTL specifications: The proposed algorithm, which is a fast, probabilistically complete, and asymptotically optimal method for generating vehicle trajectories satisfying a given LTL specification. In comparison with the two works in the literature closest to the proposed approach [106, 107], the proposed algorithm is faster owing to the proposed sampling heuristic. The work [106] constructs a RRG-based discrete transition system to represent the vehicle model. In contrast, the proposed algorithm *incrementally* constructs a tree structure, which does not require a graph search (RRG does), while retaining an asymptotic optimality property. In contrast to other randomized sampling-based approaches using a tree structure, the proposed algorithm is capable of finding finite or infinite trajectories in prefix-suffix form to satisfy LTL specifications, including specifications that lie outside of the co-safe class of specifications (will be discussed in chapter 5).

Chapter 2

Motion-planning for a Nonholonomic Vehicle

2.1 Problem Formulation

We introduce the following elements of the problem: the vehicle model, the workspace partition, and the LTL specification. We use the term *trajectory* to refer to a locus of points in the state space of the vehicle model, and the term *route* to refer to a discrete representation of a trajectory (e.g., a sequence of waypoints or regions). We discuss *route-planning* (synonymous with *route guidance*), which is the process of finding a sequence of regions in the workspace that enclose the aircraft's desired trajectory.

2.1.1 Vehicle Model

Let $\mathcal{W} \subset \mathbb{R}^2$ be a compact set, called the *workspace*, which is assumed to be a planar region of interest for the vehicle, and $v_{\max} > v_{\min} > 0$ indicate, respectively, upper and lower bounds on the vehicle speed. Let $\xi = (x, y, v, \psi) \in \mathcal{D} = \mathcal{W} \times [v_{\min}, v_{\max}] \times \mathbb{S}^1$ denote the state of the vehicle, namely, the position of the vehicle's center of mass and the magnitude and direction of its velocity vector in a prespecified Cartesian coordinate system. We denote by $\mathfrak{x}(\xi)$ the projection of $\xi \in \mathcal{D}$ on the set \mathcal{W} . We consider a vehicle kinematic model described by the differential equations

$$\dot{x}(t) = v(t) \cos \psi(t), \quad \dot{y}(t) = v(t) \sin \psi(t), \quad \dot{v}(t) = u_1(t), \quad \dot{\psi}(t) = \frac{u_2(t)}{v(t)}, \quad (2.1)$$

2.1. PROBLEM FORMULATION

where u_1 and u_2 (the tangential and lateral accelerations, respectively) are the control inputs. We assume that the set of admissible control input values is the compact domain

$$U := \left\{ (u_1, u_2) \in \mathbb{R}^2 \mid \frac{u_1^2}{a^2} + u_2^2 \rho^2 \leq 1 \right\}, \quad (2.2)$$

where $a_{\max}, \rho > 0$ are prespecified. Let \mathcal{U} be the set of all piecewise continuous functions of t defined on finite intervals that take values in U . For any $u \in \mathcal{U}$, and initial state $\xi_0 \in \mathcal{D}$, the state trajectory $\xi(t; \xi_0, u)$, $t \in [0, t_f]$, obtained by integrating (2.1) is called an *admissible state trajectory*. Note that a_{\max} is an upper bound on the magnitude of the tangential acceleration, and ρ is the minimum radius of turn at unit speed. For computational reasons to be clarified later, we assume $\rho v_{\min} > 3$.

2.1.2 Workspace partition

Consider a partition of \mathcal{W} into convex polytopic subregions called *cells*. The intersection of any two cells is either empty, or a single vertex, or a finite-length segment that lies on the boundaries of both cells. We denote by $N^C \in \mathbb{Z}_+$ the number of cells, and by $c[i] \subset \mathcal{W}$ the subregion associated with the i^{th} cell, for each $i = 1, \dots, N^C$. Therefore, $\cup_{i=1}^{N^C} c[i] = \mathcal{W}$. We associate with this partition an undirected graph $\mathcal{G} := (V, E)$ such that each vertex of \mathcal{G} is uniquely associated with a cell, and each edge of \mathcal{G} is uniquely associated with a pair of geometrically adjacent cells. We assume “4-connectivity,” i.e., two cells are considered geometrically adjacent if their intersection is a finite-length segment. We denote by $\text{cell}(v)$ the element of $\{c[i]\}_{i=1}^{N^C}$ associated with the vertex $v \in V$. A *path* \mathbf{v} in \mathcal{G} is a sequence (v_0, v_1, \dots) of vertices, such that $v_0, v_k \in V$, and $(v_{k-1}, v_k) \in E$, for each $k \in \mathbb{N}$. The number of vertices in a path is called its *length*. According to the preceding definition, a path in \mathcal{G} can contain cycles. We denote by $\mathcal{L}_{\mathcal{G}}$ the collection of all paths in \mathcal{G} . Note that the paths in $\mathcal{L}_{\mathcal{G}}$ are associated with vehicle routes described by a sequence of successively adjacent cells.

2.1. PROBLEM FORMULATION

For every $t_f \in \mathbb{R}_+$, $\xi_0 \in \mathcal{D}$, and $u \in \mathcal{U}$, we define the \mathcal{G} -trace of the trajectory $\xi(t; \xi_0, u)$, $t \in [0, t_f]$ as the path $tr(\xi, \mathcal{G}) = (v_0, v_1, \dots, v_P) \in \mathcal{L}_{\mathcal{G}}$ of minimal length such that

1. $x(\xi(0; \xi_0, u)) \in \text{cell}(v_0)$,
2. There exists a positive and strictly increasing sequence $\{t_0, t_1, \dots, t_P\}$ with $t_0 = 0$, $t_P = t_f$, and

$$x(\xi(t; \xi_0, u)) \in \text{cell}(v_k), \quad t \in [t_{k-1}, t_k], \quad \text{for each } k = 1, 2, \dots, P. \quad (2.3)$$

The preceding definition applies also to trajectories defined over $[0, \infty)$, where the aforesaid increasing sequence is infinite, and the \mathcal{G} -trace is infinitely long. We denote by $\mathcal{L}_{\Gamma}(\xi_0) \subseteq \mathcal{L}_{\mathcal{G}}$ the collection of \mathcal{G} -traces of all admissible trajectories for every $t_f \in \mathbb{R}_+$, including those defined over $[0, \infty)$. Informally, the path $tr(\xi, \mathcal{G})$ is associated with the sequence of cells that defines a “channel” in \mathcal{W} , such that the curve $x(\xi(t))$, $t \in [0, t_f]$, lies within this channel. The curve $x(\xi(t))$ and the trajectory $\xi(t)$ are said to *traverse* this channel of cells.

2.1.3 LTL_{-X} specifications

Linear temporal logic is a convenient formal language to express specifications on the behavior of a system over time. As is common in the literature [87], we use a restricted version of LTL, namely, LTL_{-X}, which does not involve the *next* operator. The choice of LTL_{-X} instead of LTL is for simplicity of exposition of the proposed work. A brief overview of LTL_{-X} is provided below; the reader is referred to the literature [2, 87] for further details.

The LTL_{-X} syntax involves operators \neg (negation), \vee (disjunction) and \triangleright (*until*). Let $\Lambda = \{\lambda_k\}_{k=0}^{N^R}$, with $N^R \in \mathbb{Z}_{\geq 0}$ be a prespecified set of atomic propositions. A LTL_{-X} formula over Λ is recursively defined as follows:

1. Every atomic proposition $\lambda_k \in \Lambda$ is a LTL_{-X} formula.

2.1. PROBLEM FORMULATION

2. If ϕ_1 and ϕ_2 are LTL_{-X} formulae, then $\neg\phi_1$, $(\phi_1 \vee \phi_2)$, and $(\phi_1 \triangleright \phi_2)$ are also LTL_{-X} formulae.

Let ϕ_1 and ϕ_2 be LTL_{-X} formulae. The formula $(\phi_1 \triangleright \phi_2)$ means that ϕ_2 eventually becomes true and ϕ_1 remains true until ϕ_2 becomes true. The operators \wedge (conjunction), \Rightarrow (implication), \Leftrightarrow (equivalence) are defined as is standard in propositional logic. The temporal operators \diamond (*eventually*), and \square (*always*) are defined as follows:

$$\diamond\phi_1 := (\phi_1 \vee \neg\phi_1) \triangleright \phi_1, \quad \square\phi_1 := \neg(\diamond\neg\phi_1).$$

A word $\bar{\omega} = (\omega_0, \omega_1, \dots)$ is a sequence such that $\omega_i \in 2^\Lambda$ for each $i = 0, 1, \dots$ where 2^Λ denotes the power set of Λ . For $i, j \in \mathbb{Z}_{\geq 0}$, $j \geq i$, we denote by ω_i^j the word $(\omega_i, \omega_{i+1}, \dots, \omega_j)$. The *satisfaction* of a LTL_{-X} formula ϕ by the word $\bar{\omega}$ is denoted by $\bar{\omega} \models \phi$, and it is recursively defined as follows:

1. $\omega \models \lambda_k$ if $\lambda_k \in \omega_0$, and $\omega \not\models \lambda_k$ if $\lambda_k \notin \omega_0$.
2. $\omega \models \neg\phi$ if $\omega \not\models \phi$.
3. $\omega \models (\phi_1 \vee \phi_2)$ if $\omega \models \phi_1$ or $\omega \models \phi_2$.
4. $\omega \models (\phi_1 \triangleright \phi_2)$ if there exists $i \geq 0$ such that $\omega_i^\infty \models \phi_2$ and for every $j < i$, $\omega_j^i \models \phi_1$.

We assume a finite number of regions of interest in the workspace labeled as $\lambda_1, \dots, \lambda_{N^R}$. Each λ_k is an atomic proposition defined by a set membership relation in \mathcal{D} of the form

$$\lambda_k \equiv x(\xi) \in \cup_{i \in \varsigma_k} c[i], \quad \text{for each } k = 1, \dots, N^R. \quad (2.4)$$

where $\varsigma_k \subseteq \{1, \dots, N^C\}$ is prespecified for each $k = 1, \dots, N^R$. Each region of interest is assumed to be a (possibly disconnected) union of cells. Each path $\mathbf{v} = (v_0, v_1, \dots) \in \mathcal{L}_{\mathcal{G}}$ defines a word $\bar{\omega}(\mathbf{v}) = (\omega_0, \omega_1, \dots)$, where

$$\omega_\ell := \{\lambda_k \mid \text{cell}(v_\ell) \subseteq \cup_{i \in \varsigma_k} c[i]\}. \quad (2.5)$$

The path \mathbf{v} is said to satisfy a LTL_{-X} formula ϕ if $\bar{\omega}(\mathbf{v}) \models \phi$. The main problem of interest in this paper is the following.

Problem 1. *Given a LTL_{-X} formula ϕ over Λ , and $\xi_0 \in \mathcal{D}$, determine a collection of paths $\mathcal{L}_{\Gamma\phi} \subseteq \mathcal{L}_{\Gamma}(\xi_0)$ such that every path in $\mathcal{L}_{\Gamma\phi}$ satisfies the formula ϕ .*

The “channel” of cells associated with every path in the collection $\mathcal{L}_{\Gamma\phi}$ can be traversed by an admissible state trajectory. Furthermore, every path in $\mathcal{L}_{\Gamma\phi}$ also satisfies the specification ϕ . The collection $\mathcal{L}_{\Gamma\phi}$ therefore represents, loosely speaking, an equivalence class of admissible state trajectories that satisfy the specification ϕ . The computation of $\mathcal{L}_{\Gamma\phi}$ is desirable because every route that belongs to $\mathcal{L}_{\Gamma\phi}$ is guaranteed to be compatible with vehicle dynamical constraints. As we discuss in Section 2.3, the solution of Problem 1 immediately leads to a route-planning algorithm.

The computation of $\mathcal{L}_{\Gamma\phi}$ is challenging because $\mathcal{L}_{\Gamma}(\xi_0)$ is difficult to compute, and involves discrete abstraction of the continuous system Γ . Previously [4, 98], feedback control laws have been designed to construct a language equivalent discrete abstraction, such that $\mathcal{L}_{\Gamma} = \mathcal{L}_{\mathcal{G}}$. However, the underlying assumption therein is that the vehicle model is completely controllable in the presence of workspace constraints (e.g. obstacles). For the vehicle model considered in this paper, this controllability assumption is not true [102].

To solve Problem 3 in the light of the preceding observations, we propose a new approach based on the so-called lifted graph, which we discuss next.

2.2 Lifted Graph

The proposed approach to the solution of Problem 3 and route-planning relies on traversability analysis and assignment of transition costs to successions of edges in the graph \mathcal{G} . To this end, we adopt and modify the idea of lifted graphs [102] as follows. For $H \in \mathbb{Z}_{\geq 0}$, let

$$V_H := \left\{ (v_0, \dots, v_H) : (v_{k-1}, v_k) \in E \text{ and } v_k \neq v_m \Leftrightarrow k \neq m, \text{ for } k, m \in \{1, \dots, H\} \right\}.$$

2.2. LIFTED GRAPH

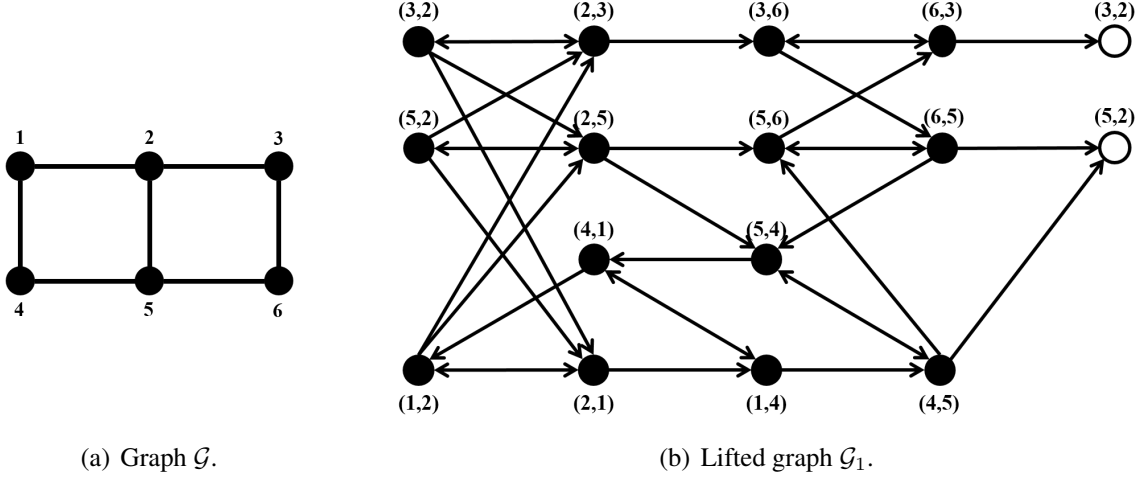


Figure 2.1: Example of a lifted graph (the vertices $(3, 2)$ and $(5, 2)$ of the lifted graph are drawn twice for clarity of the diagram).

Every element $\mathbf{i} \in V_H$ is an ordered $(H + 1)$ -tuple of unique elements of V . Per the notation used by Cowlagi & Tsiotras [102], we denote by $[\mathbf{i}]_k$ the k^{th} element of \mathbf{i} , and by $[\mathbf{i}]_k^m$ the tuple $([\mathbf{i}]_k, [\mathbf{i}]_{k+1}, \dots, [\mathbf{i}]_m)$, where $1 \leq k < m \leq H + 1$. Let E_H be a set of pairs $(\mathbf{i}, \mathbf{j}) \in V_H \times V_H$, such that $[\mathbf{i}]_k = [\mathbf{j}]_{k-1}$, for each $k = 2, \dots, H + 1$. According to this notation, $V_0 = V$ and $E_0 = E$. Note that, for $H \geq 1$, each vertex in V_H defines an *edge* in E_{H-1} . if H is larger than the length of the longest simple path in \mathcal{G} , then V_H is empty. The *lifted graph* \mathcal{G}_H is defined as the directed graph whose vertex and edge sets are, respectively, V_H and E_H . Figure 2.1 illustrates an example of a lifted graph for $H = 1$.

With the exception of paths that contain cycles of length $H + 1$ or less, every path $\mathbf{v} = (v_0, v_1, \dots)$ in the graph \mathcal{G} can be uniquely mapped to a path $\bar{\mathbf{i}} := (\mathbf{i}_0, \mathbf{i}_1, \dots)$ in the lifted graph \mathcal{G}_H , where $\mathbf{i}_k := (j_k, j_{k+1}, \dots, j_{k+H}) \in V_H$ for each $k \in \mathbb{N}$. We denote this map by $b_0^H : \mathcal{L}_{\mathcal{G}} \rightarrow \mathcal{L}_{\mathcal{G}_H}$, where $\mathcal{L}_{\mathcal{G}_H}$ is the collection of all paths in \mathcal{G}_H . Therefore, $\bar{\mathbf{i}} = b_0^H(\mathbf{v})$ and $\mathbf{v} = b_H^0(\bar{\mathbf{i}})$. For every integer $H \geq 0$, the map b_0^H is surjective and invertible and b_H^H is the identity map. As a first step towards the solution of Problem 1, we associate certain reachability properties of the vehicle model with edge transitions in the lifted graph \mathcal{G}_H . In what follows, we assume $H \geq 1$.

2.2.1 Edge Transition Costs in \mathcal{G}_H

To characterize the collection \mathcal{L}_Γ , we define a transition cost function $g_H : E_H \rightarrow [0, \chi]$, by which we can compute costs of paths in the lifted graph \mathcal{G}_H . Here, $\chi \in \mathbb{R}_+$ is a large positive number sufficiently greater than the length of the longest simplest path in \mathcal{G} . Then we assert that a path $\mathbf{v} \in \mathcal{L}_\mathcal{G}$ belongs to the collection $\mathcal{L}_\Gamma(\xi_0)$ if and only if any subpath of $b_0^H(\mathbf{v})$ of length less than χ has cost less than χ .

Consider an edge $(\mathbf{i}, \mathbf{j}) \in E_H$ in the lifted graph \mathcal{G}_H . Let $\mathcal{S}(\mathbf{i}) \subset \mathcal{D}$ be a set of states associated with $\mathbf{i} \in V_H$ such that $\mathbf{x}(\mathcal{S}(\mathbf{i})) \subseteq \text{cell}([\mathbf{i}]_1) \cap \text{cell}([\mathbf{i}]_2)$. Thus, the projections on \mathcal{W} of the elements in $\mathcal{S}(\mathbf{i})$ lie on the boundary between the first and second cells corresponding to the vertices of V that constitute the ordered H -tuple \mathbf{i} . Next let $\mathcal{Q}(\mathbf{j}) \subset \mathcal{D}$ be such that $\mathbf{x}(\mathcal{Q}(\mathbf{j})) \subseteq \text{cell}([\mathbf{j}]_1) \cap \text{cell}([\mathbf{j}]_2)$ and for every $\xi_q \in \mathcal{Q}(\mathbf{j})$ there exists a finite traversal time t_q and an admissible control input $u_q \in \mathcal{U}$ such that $\text{tr}(\xi(\cdot; \xi_q, u_q), \mathcal{G}) = b_{H-1}^0(\mathbf{j})$. Informally, $\mathcal{Q}(\mathbf{j})$ is the set of all states whose position components lie on the boundary between the first and second cells of \mathbf{j} , and such that the traversal of the geometric region defined by the cells associated with \mathbf{j} is possible from any initial state within $\mathcal{Q}(\mathbf{j})$. Sets such as $\mathcal{Q}(\cdot)$ are called *backward reachable sets* or *target sets* [108].

Next, let \mathcal{R}_i be a reachability map associated with the states in sets $\mathcal{S}(\mathbf{i})$, defined by

$$\mathcal{R}_i(\xi_s) := \left\{ \xi_t \in \mathcal{D} \mid \xi_t \in \bigcup_{t \in \mathbb{R}_+} \bigcup_{u \in \mathcal{U}_t} \xi(t; \xi_s, u), \text{ and } \left(\bigcup_{\tau \in [0, t]} \mathbf{x}(\xi(\tau; \xi_s, u)) \right) \cap (\mathcal{W} \setminus \text{cell}([\mathbf{i}]_2)) = \emptyset \right\}, \quad (2.6)$$

where $\xi_s \in \mathcal{S}(\mathbf{i})$. Informally, $\mathcal{R}_i(\xi_s)$ is a *forward reachable set* of all states that can be reached from ξ_s by trajectories whose projections on \mathcal{W} always remain within the $\text{cell}([\mathbf{i}]_2)$, i.e. the region defined by the second cell of \mathbf{i} . Finally, define $\hat{\mathcal{S}}(\mathbf{i}, \mathbf{j}) := \{\xi_s \in \mathcal{S}(\mathbf{i}) : \mathcal{R}_i(\xi_s) \cap \mathcal{Q}(\mathbf{j}) \neq \emptyset\}$.

Now consider a finite-length path $\mathbf{v} = (v_0, v_1, \dots, v_P) \in \mathcal{L}_\mathcal{G}$ with no cycles of length less than or equal to $H + 1$, and an initial vehicle state $\xi_0 \in \mathcal{D}$, with $\mathbf{x}(\xi_0) \in \text{cell}(v_0) \cap \text{cell}(v_1)$.

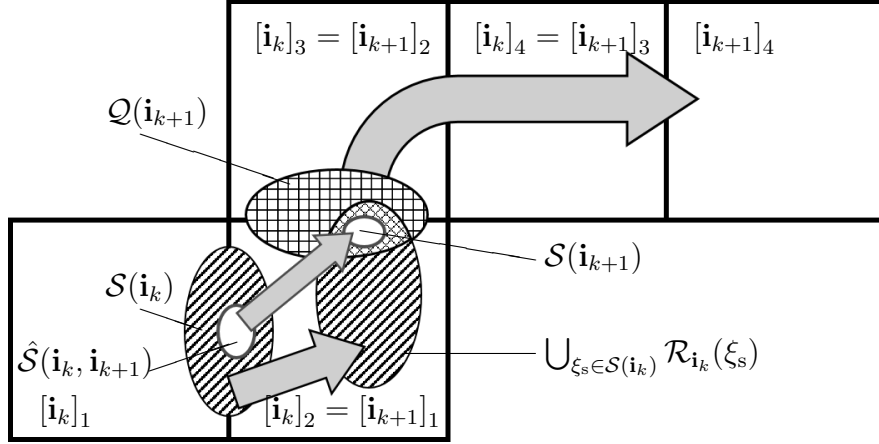


Figure 2.2: Conceptual illustration of the sets \mathcal{R} , $\mathcal{Q}(\cdot)$ and $\mathcal{S}(\cdot)$ for $H = 2$. The large arrows indicate the existence of admissible state trajectories as described in the text preceding (2.6).

For each $k \in \mathbb{N}$, let $\mathbf{i}_k := (v_k, \dots, v_{k+H})$. Clearly, $(\mathbf{i}_k, \mathbf{i}_{k+1}) \in E_H$. We iteratively define g_H and the set association $\mathcal{S}(\cdot)$ as follows:

$$\mathcal{S}(\mathbf{i}_{k+1}) := \bigcup_{\xi_s \in \hat{\mathcal{S}}(\mathbf{i}_k, \mathbf{i}_{k+1})} (\mathcal{R}_{\mathbf{i}_k}(\xi_s) \cap \mathcal{Q}(\mathbf{i}_{k+1})), \quad (2.7)$$

$$g_H(\mathbf{i}_k, \mathbf{i}_{k+1}) := \begin{cases} \chi, & \text{if } \mathcal{S}(\mathbf{i}_{k+1}) = \emptyset, \\ 1, & \text{otherwise,} \end{cases} \quad (2.8)$$

where $\mathcal{S}(\mathbf{i}_0) = \xi_0$. The H -cost of the path $\mathbf{v} \in \mathcal{L}_{\mathcal{G}}$ is defined as

$$\mathcal{J}_H(\mathbf{v}) := H + \sum_{k=0}^{P-H} g_H(\mathbf{i}_k, \mathbf{i}_{k+1}). \quad (2.9)$$

Figure 2.2 illustrates these concepts. Note first that the H -cost of a path $\mathbf{v} \in \mathcal{L}_{\mathcal{G}}$ depends on the initial state. The collection of sets $\mathcal{S}(\mathbf{i}_k)$ indicate the possible vehicle states from which traversal of the channel associated with the path \mathbf{v} is possible. Second, note that due to the recursive nature of (2.7)–(2.9), it is beneficial to compute edge transition costs in \mathcal{G}_H in conjunction with a search algorithm for finding a desired path in $\mathcal{L}_{\mathcal{G}}$.

In this context, notice that a serious issue arises. On the one hand, the computations

2.2. LIFTED GRAPH

of the sets $\mathcal{R}(\cdot)$, $\mathcal{Q}(\cdot)$, and $\mathcal{S}(\cdot)$ are time-intensive and unsuitable for real-time implementation. On the other hand, the edge transition costs in (2.7)–(2.9) must be computed simultaneously with the real-time route-planning algorithm (see Section 2.3.1) that searches for a path in the lifted graph \mathcal{G}_H . To resolve this issue, we propose a technique that relegates the time-intensive computations to an offline preprocessing stage, and yet allows for conservative approximations of the sets $\mathcal{R}(\cdot)$, $\mathcal{Q}(\cdot)$, and $\mathcal{S}(\cdot)$ to be determined online (based on the preprocessed results) within the route-planning algorithm. This technique is discussed in Section 2.4. Here, we state an important result that characterizes the collection $\mathcal{L}_\Gamma(\xi_0)$ of \mathcal{G} -traces of all admissible vehicle trajectories.

Proposition 1. *Let $\mathbf{v} = (v_0, \dots, v_P)$ be a path in $\mathcal{L}_\mathcal{G}$ with length less than χ and with no cycles of length less than or equal to $H + 1$, and let $\xi_0 \in \mathcal{D}$ be prespecified such that $\mathbf{x}(\xi_0) \in \text{cell}(v_0) \cap \text{cell}(v_1)$. Then $\mathbf{v} \in \mathcal{L}_\Gamma(\xi_0)$ if and only if $\mathcal{J}_H(\mathbf{v}) < \chi$.*

Proof. Define $\mathbf{i}_k := (v_k, \dots, v_{k+H})$, for each $k \in \{0, \dots, P - H\}$. First, suppose that $\mathcal{J}_H(\mathbf{v}) < \chi$. By Eqns. (2.8) and (2.9), it follows that, for each $k \in \{0, \dots, P - H\}$, $g_H(\mathbf{i}_k, \mathbf{i}_{k+1}) = 1$, and that $\mathcal{S}(\mathbf{i}_{k+1})$ is nonempty. By Eqn. (2.7), for every state $\xi_s \in \mathcal{S}(\mathbf{i}_{k+1})$, there exists $\text{pre}(\xi_s) \in \hat{\mathcal{S}}(\mathbf{i}_k, \mathbf{i}_{k+1}) \subseteq \mathcal{S}(\mathbf{i}_k)$ such that $\xi_s \in \mathcal{R}_{\mathbf{i}_k}(\text{pre}(\xi_s))$.

In particular, $\mathcal{S}(\mathbf{i}_{P-H})$ is nonempty, and, by Eqn. (2.7), $\mathcal{Q}(\mathbf{i}_{P-H})$ is nonempty. By definition, for any state $\xi_{P-H} \in \mathcal{Q}(\mathbf{i}_{P-H})$, there exist $t_{P-H} \in \mathbb{R}_+$ and a control input $u_{P-H} \in \mathcal{U}_{t_{P-H}}$ such that

$$\mathbf{x}(\xi(t; \xi_{P-H}, u_{P-H})) \in \cup_{\ell=1}^{H+1} \text{cell}([I_{P-H}]_\ell) \quad (2.10)$$

Then we iteratively define $\xi_k := \text{pre}(\xi_{k+1})$ for each $k = P - H, \dots, 0$. Note that $\xi_{k+1} \in \mathcal{R}_{I_k}(\xi_k)$. By definition in Eqn. (2.6), there exist $t_k \in \mathbb{R}_+$ and $u_k \in \mathcal{U}_{t_k}$ such that $\xi(t; \xi_k, u_k) = \xi_{k+1}$, and

$$\mathbf{x}(\xi(t; \xi_k, u_k)) \in \text{cell}([\mathbf{i}_k]_2), \quad t \in [0, t_k]. \quad (2.11)$$

2.3. PRODUCT TRANSITION SYSTEM

Note that $\xi_k \in \mathcal{S}(\mathbf{i}_k)$, which implies that $\xi_0 \in \mathcal{S}(\mathbf{i}_0) = \xi_0$. Now define the concatenated trajectory

$$\xi^*(t) := \xi(t; \xi_k, u_k), \quad t \in [\tau_k, \tau_{k+1}], \quad (2.12)$$

where $\tau_0 = 0$ and $\tau_{k+1} := \tau_k + t_k$, for each $k = 0, \dots, P - H$. By Eqns. (2.10) and (2.11), it follows that $\mathbf{v} = tr(\xi^*, \mathcal{G})$, and, by consequence, that $\mathbf{v} \in \mathcal{L}_\Gamma(\xi_0)$.

To prove the converse, suppose that $\mathbf{v} \in \mathcal{L}_\Gamma(\xi_0)$. By definition, $\mathbf{x}(\xi(0; \xi_0, u)) \in \text{cell}(v_0)$ and $\mathbf{x}(\xi(t_f; \xi_0, u)) \in \text{cell}(v_P)$. Define for each $k \in \{0, \dots, P - H\}$,

$$\begin{aligned} \tau_k &:= \max_{\tau_k \in [0, t_f]} \{ \tau_k : \mathbf{x}(\xi(\tau_k; \xi_0, u)) \in \text{cell}([\mathbf{i}_k]_1) \cap \text{cell}([\mathbf{i}_k]_2) \}, \\ \xi_k^s &:= \xi(\tau_k; \xi_0, u), \quad u_k := u|_{[\tau_k, \tau_{k+1}]}. \end{aligned}$$

The existence of τ_k is guaranteed by continuity of ξ and by (2.3). It is easy to see that $\tau_0 = 0$, $\xi_k^s = \xi_0$, and that for each $k \in \{1, \dots, P - H\}$, $\xi_{k+1}^s \in \mathcal{R}_{\mathbf{i}_k}(\xi_k^s)$. By definition of the sets $\mathcal{Q}(\cdot)$, it is also clear that $\xi_{k+1}^s \in \mathcal{Q}(\mathbf{i}_{k+1})$. Therefore, $\xi_{k+1}^s \in \mathcal{S}(k + 1)$, and by Eqns. (2.7) and (2.8), $g_H(\mathbf{i}_k) = 1$ for each $k \in \{1, \dots, P - H\}$. It follows that $\mathcal{J}_H(\mathbf{v}) < \chi$. \square

Proposition 1 asserts that the channel of cells associated with any path in $\mathcal{L}_\mathcal{G}$ is traversable by an admissible state trajectory of Γ if and only if the H -cost of this path is less than χ (assuming that the number of vertices in this path is less than χ).

2.3 Product Transition System

The lifted graph \mathcal{G}_H and the associated edge transition cost computations define a finite state transition system that represents the vehicle dynamical model. To find paths in \mathcal{G}_H that satisfy a given specification ϕ , we construct and search a so-called *product transition system* as discussed below. Figure 3.1 illustrates the proposed route-planning algorithm. Recall that the connection between the given LTL specification and the UAV route is via (2.4), which

2.3. PRODUCT TRANSITION SYSTEM

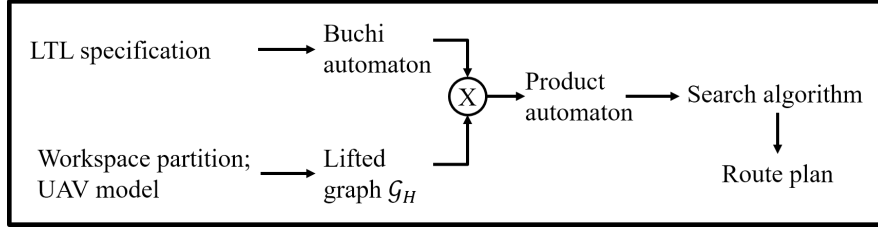


Figure 2.3: Conceptual illustration of the proposed route-planning approach.

associates propositions in the LTL specifications with regions in the workspace.

It is known [88, 89] that every LTL formula can “represented” by a finite state transition system. Precisely, every LTL formula ϕ over the alphabet Λ is associated with a *Büchi automaton* \mathcal{B}_ϕ with input alphabet 2^Λ . The collection of accepting runs of \mathcal{B}_ϕ is exactly the collection of infinite strings over Λ that satisfy ϕ . In the context of UAV route-planning, admissible transition sequences in \mathcal{B}_ϕ are associated with routes that satisfy the given LTL specification. Algorithms for translating a LTL formula to the associated Büchi automaton are available [89–91]. The reader interested is referred to the literature [2, 109] for further details on finite state automata in general and Büchi automata in particular.

For the Büchi automaton \mathcal{B}_ϕ , we denote by S the set of states, by $\delta_{\mathcal{B}_\phi} \subseteq S \times 2^\Lambda \times S$ the transition relation between states, and by $S_0, S_f \subseteq S$, respectively, the sets of initial and accepting states. For an integer $H \geq 1$, we now define a *product transition system* $\mathcal{T}_{\phi, H} := (T, \delta_{\mathcal{T}_{\phi, H}})$ as follows:

1. The set of states of $\mathcal{T}_{\phi, H}$ is $T := S \times V_H$. For each state $\theta \in T$, we denote by $\theta|_S$ and $\theta|_{V_H}$, respectively, the projections of θ on S and V_H .
2. The transition relation of $\mathcal{T}_{\phi, H}$ is $\delta_{\mathcal{T}_{\phi, H}} \subseteq T \times 2^\Lambda \times T$ defined as the set of all triplets $(\theta_k, \omega_k, \theta_\ell)$ such that

$$(\theta_k|_S, \omega_k, \theta_\ell|_S) \in \delta_{\mathcal{B}_\phi}, \quad (\theta_k|_{V_H}, \theta_\ell|_{V_H}) \in E_H, \quad (2.13)$$

$$\omega_k = \{\lambda_i \mid \text{cell}([\theta_k|_{V_H}]_1) \subseteq \cup_{j \in \mathfrak{c}_i} c_j\}. \quad (2.14)$$

2.3. PRODUCT TRANSITION SYSTEM

Thus, transitions in the product system are associated with transitions in the Büchi automaton *and* with transitions in the lifted graph \mathcal{G}_H . The relationship between transitions in the product system and the UAV's workspace is defined by (3.3).

A *run* of $\mathcal{T}_{\phi,H}$ is a sequence $\Theta = (\theta_0, \theta_1, \dots)$ such that $\theta_k \in T$ for each $k \in \mathbb{Z}_{\geq 0}$, and $(\theta_k, \omega_k, \theta_{k+1}) \in \delta_{\mathcal{T}_{\phi,H}}$, with ω_k as defined in (3.3). We denote by $\Theta|_S = (\theta_0|_S, \theta_1|_S, \dots)$ and $\Theta|_{V_H} = (\theta_0|_{V_H}, \theta_1|_{V_H}, \dots)$, respectively, the projections of Θ on S and V_H . Note that $\Theta|_{V_H} \in \mathcal{L}_{\mathcal{G}_H}$, and therefore $b_H^0(\Theta|_{V_H}) \in \mathcal{L}_{\mathcal{G}}$.

Similar to previously reported approaches in the literature [87], we restrict attention to runs of $\mathcal{T}_{\phi,H}$ of a “prefix-suffix” form $\Theta = (\Theta_p, \Theta_s, \Theta_s, \dots)$. Here, the “suffix” run $\Theta_s = (\theta_{f,1}, \theta_{s,1}, \dots, \theta_{s,N}, \theta_{f,2})$, which is repeated infinitely often in Θ , is a finite sequence such that $\theta_{f,1}, \theta_{f,2} \in S_f \times V_H$, and $\theta_{s,n} \in S \times V_H$, for $n = 1, \dots, N$. The “prefix” run $\Theta_p = (\theta_0, \dots, \theta_M)$ is a finite sequence such that $\theta_0 \in S_0 \times V_H$ and $(\theta_M, \omega_M, \theta_f) \in \delta_{\mathcal{T}_{\phi,H}}$. Now we state the main result of this paper as follows.

Theorem 1. *Let $\Theta = (\theta_0, \theta_1, \dots)$ be a run of $\mathcal{T}_{\phi,H}$.*

$$\begin{aligned} & \text{If } \mathcal{J}_H(b_H^0(\Theta_p|_{V_H})) < \chi \quad \text{and} \quad \mathcal{J}_H(b_H^0(\Theta_s|_{V_H})) < \chi, \\ & \text{then } b_H^0(\Theta|_{V_H}) \in \mathcal{L}_{\Gamma\Phi}. \end{aligned}$$

Conversely, for every path $\mathbf{v} \in \mathcal{L}_{\Gamma\Phi}$ with no cycles of length less than or equal to $H + 1$, there exists a run Θ of $\mathcal{T}_{\phi,H}$, such that $b_H^0(\Theta|_{V_H}) = \mathbf{v}$.

Proof. By Prop. 1,

$$b_H^0(\Theta_p|_{V_H}) \in \mathcal{L}_{\Gamma}(\xi_0), \quad b_H^0(\Theta_s|_{V_H}) \in \mathcal{L}_{\Gamma}(\xi_0),$$

which implies that $b_H^0((\Theta_p, \Theta_s, \Theta_s, \dots)|_{V_H}) \in \mathcal{L}_{\Gamma}(\xi_0)$. By (2.5), (3.2), and (3.3), the path $\mathbf{v} := b_H^0(\Theta|_{V_H}) \in \mathcal{L}_{\mathcal{G}}$ defines the word $\bar{\omega}(\mathbf{v}) = (\omega_0, \omega_1, \dots)$. By definition of $\mathcal{T}_{\phi,H}$, $(\theta_k, \omega_k, \theta_{k+1}) \in \delta_{\mathcal{T}_{\phi,H}}$, and $(\theta_k|_S, \omega_k, \theta_{k+1}|_S) \in \delta_{\mathcal{B}_{\phi}}$ for each $k \in \mathbb{N}$. Therefore, the word

2.3. PRODUCT TRANSITION SYSTEM

$\omega(\mathbf{v})$ is accepted by the Büchi automaton \mathcal{B}_ϕ , which in turn means that the path π satisfies the formula ϕ and, by consequence, $\mathbf{v} = b_H^0(\Theta_{\mathcal{P}}|_{V_H}) \in \mathcal{L}_{\Gamma\phi}$.

To prove the converse, consider a path $\mathbf{v} = (j_0, j_1, \dots) \in \mathcal{L}_{\Gamma\phi} \subseteq \mathcal{L}_\Gamma \subseteq \mathcal{L}_{\mathcal{G}}$ such that the length of \mathbf{v} is greater than $H + 1$ and there are no cycles of length less than or equal to $H + 1$. Because the path satisfies the formula ϕ , the word $\omega(\mathbf{v}) = (\omega_0, \omega_1, \dots)$, where ω_k is defined in Eqn. (2.5), is accepted by the Büchi automaton \mathcal{B}_ϕ . Let s_0, s_1, \dots be the states of \mathcal{B}_ϕ visited in the run associated with the input word $\overline{\omega}(\mathbf{v})$. Also, let $\mathbf{i}_k := (v_k, \dots, v_{k+1}) \in V_H$, and define $\theta_k := (s_k, \mathbf{i}_k)$ for each $k \in \mathbb{N}$. Clearly, $(\theta_k, \omega_k, \theta_{k+1}) \in \delta_{\mathcal{T}_{\phi,H}}$, and it follows that $\Theta := (\theta_0, \theta_1, \dots)$ is a run of $\mathcal{T}_{\phi,H}$, and that $\Theta|_{V_H} = b_0^H(\mathbf{v})$, which implies $b_H^0(\Theta|_{V_H}) = \mathbf{v}$. \square

2.3.1 Route-Planning Algorithm

Theorem 2 solves Problem 3 in that it precisely characterizes the collection $\mathcal{L}_{\Gamma\phi}$. Recall that $\mathcal{L}_{\Gamma\phi}$ is an equivalence class of admissible state trajectories that satisfy the specification ϕ . Following Theorem 2, it is easy to determine a specific plan for a given initial state $\xi_0 \in \mathcal{D}$ and a given LTL_{-X} specification ϕ . To do so, we execute Dijkstra's algorithm to search for a prefix and suffix run of the product transition system that satisfy the conditions given in Theorem 2. A straightforward algorithm for finding a run of a product automaton with minimum total number of vertices in the concatenation of prefix and suffix runs appears in the literature [87], which we can appropriately modify for finding a run of $\mathcal{T}_{\phi,H}$. A limitation of the proposed algorithm is that routes of length less than $H + 1$ or routes containing cycles of length less than or equal to $H + 1$ cannot be mapped to the lifted graph \mathcal{G}_H , and will therefore not be found by the proposed algorithm. However, for practical applications in aircraft route-planning this fact is unlikely to restrict the proposed algorithm. The values of H are typically small integers, whereas routes to be planned in practice are over large environments. Therefore, routes with less than H cells are unlikely to be of significance. Furthermore, the aircraft's minimum radius of turn is likely to preclude short-length cycles in the route.

2.3. PRODUCT TRANSITION SYSTEM

The only modification that needs to be made to Dijkstra’s search algorithm is that the edge transition costs in the lifted graph are computed simultaneously with the search. To do so, within the search algorithm each search vertex (state of the product transition system) is associated with an index set related to the vehicle state. The exact definition of these index sets is clarified in the next Section, where we also introduce a connectivity matrix.

This idea is illustrated with pseudo-code in Fig. 2.4. Let $\xi_0 \in \mathcal{D}$ be the given vehicle initial state and let v^S be the vertex in \mathcal{G} such that $\xi_0 \in \text{cell}(v^S)$. We define θ^S as a “dummy” start vertex in the product automaton $\mathcal{T}_{\phi,H}$ such that it is connected with zero cost to all $\theta \in T$ with $[\theta|_{V_H}]_1 = v^S$. As is usual in Dijkstra’s algorithm [58], each search vertex (state of the product transition system) is associated with a label d and a backpointer b . The *fringe* \mathcal{P} is a set of search vertices whose label can potentially be improved. The REMOVE and INSERT functions maintain a sorting order in the fringe according to the current value of the label. The modification proposed here associates the index set R with each search vertex, which is updated in Line 8 of the pseudo-code shown in Fig. 2.4. The meaning and significance of the symbol C (connectivity matrix) in Line 8 will become clear in the next Section. The description in Fig. 2.4 is an idealization, and implementations of the proposed route-planning can be significantly sped up by using search heuristics, a discussion of which is beyond the scope of this paper. The cost function (2.8) is approximated as follows, to provide transition costs in the product system $\mathcal{T}_{\phi,H}$:

$$\tilde{g}_H((\theta_k|_{V_H}, \theta_\ell|_{V_H})) := \begin{cases} \chi, & \text{if } R(\theta_\ell) = \emptyset, \\ 1, & \text{otherwise.} \end{cases}$$

The computational complexity of the proposed algorithm is the same as that of Dijkstra’s algorithm, namely, $\mathcal{O}(|\delta_{\mathcal{T}_{\phi,H}}| + |\mathcal{T}_{\phi,H}| \log |\mathcal{T}_{\phi,H}|)$ when the fringe \mathcal{P} is implemented using a Fibonacci heap [110]. Here, $\delta_{\mathcal{T}_{\phi,H}}$ is the total number of transitions in the product transition system. To express this computational complexity in terms of problem data, namely, the number of cells N^C and the number of atomic propositions N^R in the given LTL_{-X}

Proposed Label-Correcting Algorithm for Route-Planning

procedure INITIALIZE(θ_S)

- 1: $\mathcal{P} := \theta^S$, $d(\theta^S) := 0$, $R(\theta^S) := m_0$.
- 2: **for all** $\theta \in T$ **do**
- 3: $d(\theta) := \infty$

procedure MAIN

- 1: INITIALIZE(θ^S)
 - 2: **while** $\mathcal{P} \neq \emptyset$ **do**
 - 3: $\theta_k := \text{REMOVE}(\mathcal{P})$
 - 4: **for all** $\theta_\ell \in T$ such that there exists ω_k as in (3.3) and $(\theta_k, \omega_k, \theta_\ell) \in \delta_{\mathcal{T}_{\phi,H}}$ **do**
 - 5: **if** $d(\theta_k) + \tilde{g}_H((\theta_k|_{V_H}, \theta_\ell|_{V_H})) < d(\theta_\ell)$ **then**
 - 6: $d(\theta_\ell) := d(\theta_k) + \tilde{g}_H((\theta_k|_{V_H}, \theta_\ell|_{V_H}))$; $b(\theta_\ell) := \theta_k$
 - 7: $R(\theta_\ell) := \{p \in \mathbb{N} \mid C_{mp} = 1, \text{ for each } m \in R(\theta_k)\}$
 - 8: $\mathcal{P} := \text{INSERT}(\mathcal{P}, \theta_\ell)$
-

Figure 2.4: Pseudo-code for executing a modified form of Dijkstra’s algorithm on the product automaton.

specification, note first that that $|\mathcal{T}_{\phi,H}| = |V_H||S|$, and $|\delta_{\mathcal{T}_{\phi,H}}| \approx |E_H||S|$. The minimum number of states in the Büchi automaton \mathcal{B}_ϕ depends on the structure of ϕ ; for specifications of typical relevance in aircraft guidance applications (see Section 5.4 for illustrative examples), we consider $|S| \approx N^R + 1$, as reported in the literature [111]. Next, we note that the number of edges in the cell decomposition graph $\mathcal{G} = \mathcal{G}_0$ is of the same order as the number of cells, i.e. $E \approx 2N^C$, for the specific case of 4-connected rectangular cells considered in this paper. Next, note that $|V_H| \approx N^C 4^H$, and that $E_H \approx 4|V_H| \approx N^C 4^{H+1}$. It follows that the worst-case complexity of the proposed algorithm is $\mathcal{O}(N^T 4^{H+1} + N^T 4^H \log(N^T 4^H))$, where $N^T := N^C(N^R + 1)$.

2.4 Numerical Computation of Edge Transition Costs

In this section, we address the computation of the sets $\mathcal{R}(\cdot)$, $\mathcal{Q}(\cdot)$, and $\mathcal{S}(\cdot)$ that appear in the edge transition cost definition (2.7)–(2.9).

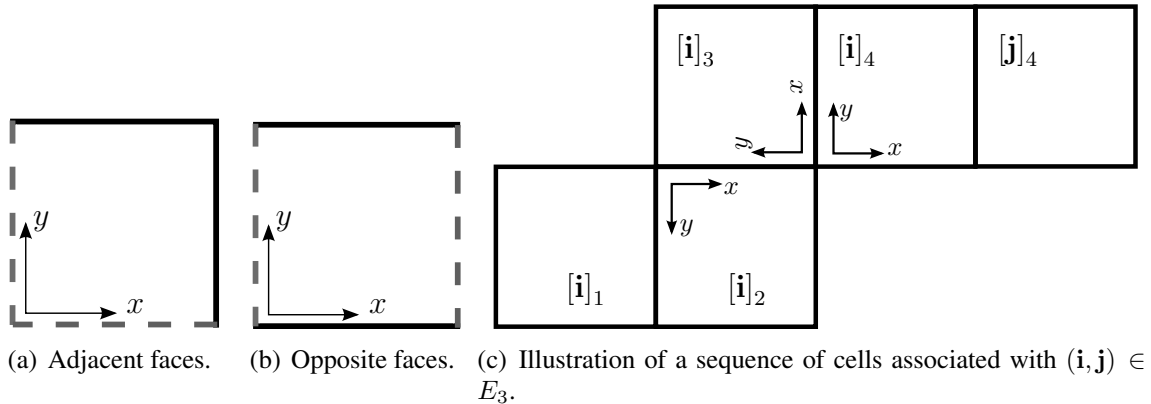


Figure 2.5: Illustration of canonical traversals (adjacent and opposite) and local coordinate axes systems. In (a) and (b), the green- and red colored dotted lines indicate, respectively, the entering and exiting faces of traversal of the cell.

Admissible state trajectories of the vehicle model in Section 2.1.1 are continuously differentiable planar curves with a speed profile, such that the curvature at any point on these curves satisfies the speed-dependent upper bound

$$\kappa_{\max} := \frac{\sqrt{a_{\max}^2 - \dot{v}^2}}{v^2 a_{\max} \rho}. \quad (2.15)$$

Recall that, due to the control input constraint (2.2), $|\dot{v}| \leq a_{\max}$, which ensures that the previous expression for κ_{\max} is real. Based on this observation, and previously reported geometric analysis of curvature-bounded paths [112], we discuss a method to numerically compute the edge transition costs defined in (2.7)–(2.9). In the analysis that follows, we assume that the workspace has been partitioned using square cells of uniform size. Without loss of generality, we assume that the size of each side of any cell is 1 unit.

First, recall that each edge $(\mathbf{i}, \mathbf{j}) \in E_H$ is associated with a sequence of $H + 2$ successively adjacent cells, namely, the sequence $(\text{cell}([\mathbf{i}]_1), \dots, \text{cell}([\mathbf{i}]_{H+1}), \text{cell}([\mathbf{j}]_{H+1}))$ as illustrated in Fig. 2.5(c). Each cell in this sequence, called a *tile* [102], admits either traversal across adjacent faces (e.g. $\text{cell}([\mathbf{i}]_2)$ and $\text{cell}([\mathbf{i}]_3)$ in Fig. 2.5(c)), or traversal across opposite faces (e.g. $\text{cell}([\mathbf{i}]_4)$ in Fig. 2.5(c)). We consider canonical forms of these cell traversal types, and attach local, right-handed coordinate axes systems as shown in Figs. 2.5(a) and 2.5(b).

Similarly, local coordinate axes systems are attached to each cell such that (a) for traversal across adjacent faces, the x -axis coincides with the “exiting” face, and (b) for either type of traversal, the y -axis coincides with the “entering” face, as shown in Fig. 2.5(c). Note that each local axes system can be transformed via rigid rotations and/or reflections to either of the canonical forms shown in Figs. 2.5(a) and 2.5(b).

The advantage of attaching these local coordinate axes systems to each cell is that, for $k = 1, \dots, H$, for every state $\xi_s \in \text{cell}([\mathbf{i}]_k) \cap \text{cell}([\mathbf{i}]_{k+1})$, the local coordinates of $\mathbf{x}(\xi_s)$ are of the form $(0, w)$, where $0 \leq w \leq 1$. Consequently, the local coordinates of the states in the previously introduced sets $\mathcal{Q}(\cdot)$ and $\mathcal{S}(\cdot)$ reside in the parallelepiped $\mathfrak{S} := [0, 1] \times [-\frac{\pi}{2}, \frac{\pi}{2}] \times [v_{\min}, v_{\max}]$. This observation is crucial for developing a concise representation of (approximations of) the forward- and backward reachable sets involved in (2.7)–(2.9), and consequently enable fast online computations thereof.

The preprocessing stage of the proposed approach is summarized as follows. First, we generate a library of tiles for each relevant value of $H = 1, 2, \dots$. Next, for each of these tiles in this library, we consider two copies of the parallelepiped $\mathfrak{S} := [0, 1] \times [-\frac{\pi}{2}, \frac{\pi}{2}] \times [v_{\min}, v_{\max}]$ associated with, respectively, the second and third cells in the tile. These copies are denoted, respectively, by \mathfrak{S}_1 and \mathfrak{S}_2 . Next, we compute numerical approximations of the sets of the sets $\mathcal{R}(\cdot)$, $\mathcal{Q}(\cdot)$, and $\mathcal{S}(\cdot)$ using geometric analysis of curvature-bounded planar curves. Next, we partition the parallelepiped \mathfrak{S} into uniformly sized regions. Finally, we compute intersections or inclusions of these regions with the sets $\mathcal{R}(\cdot)$, $\mathcal{Q}(\cdot)$, and $\mathcal{S}(\cdot)$, as appropriate, to determine a connectivity matrix of regions within \mathfrak{S}_1 to those within \mathfrak{S}_2 . This connectivity matrix is stored and looked up during online computations of edge transition costs in \mathcal{G}_H . In what follows, we provide additional details of each of the steps outlined above.

2.4. NUMERICAL COMPUTATION OF EDGE TRANSITION COSTS

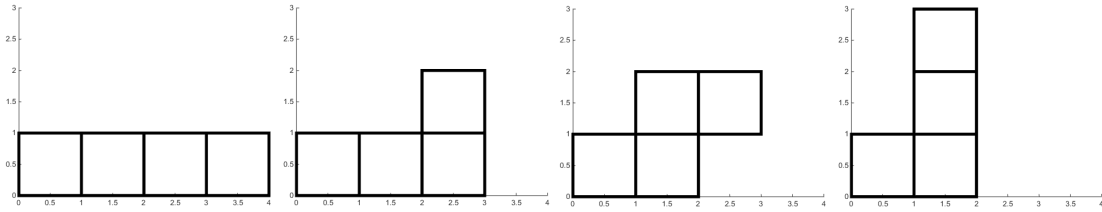


Figure 2.6: Tile library for $H = 2$, auto-generated using a MATLAB[®] script.

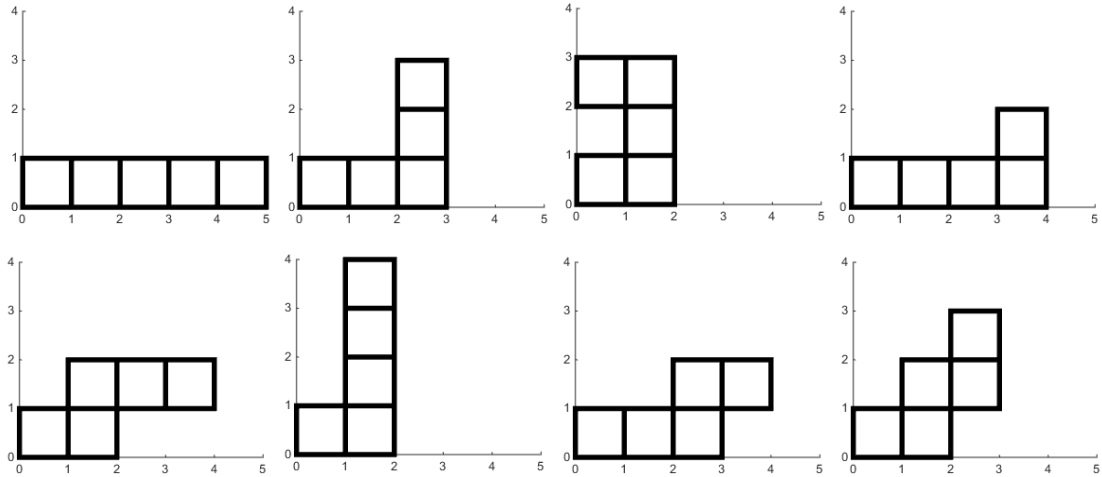


Figure 2.7: Tile library for $H = 3$, auto-generated using a MATLAB[®] script.

Library of Tiles

Recall that for each $H \geq 1$, the number of cells in any tile is $H + 2$. It is straightforward to generate a library of tiles for each value of H by enumerating all possible permutations of successive traversal types. During real-time computations, the sequence of cells associated with any edge in \mathcal{G}_H can be uniquely mapped to a tile in this library via rigid transformations. The libraries of all tiles (unique up to rigid transformations) for $H = 2$ and $H = 3$ are shown in Figs. 2.6 and 2.7 respectively.

Geometric Computation of Forward- and Backward Reachable Sets

Computations of the backward reachable set $\mathcal{Q}(\cdot)$ can be performed using previously reported analysis [112] of curvature-bounded curves in rectangular channels similar to the

regions enclosed by tiles. The details of this analysis are available in the literature [112, 113], and are therefore omitted from this paper. Here, it suffices to note that, for the tile associated with an edge $(\mathbf{i}, \mathbf{j}) \in E_H$ and a prespecified $\kappa > 0$, the results of this analysis are pointwise numerical values of four piecewise continuous functions $\underline{\alpha}_1, \bar{\alpha}_1, \underline{\alpha}_2, \bar{\alpha}_2 : [0, 1] \rightarrow [-\frac{\pi}{2}, \frac{\pi}{2}]$ such that:

1. For every point with coordinates $(0, w)$ on the cell boundary $\text{cell}([\mathbf{i}]_1) \cap \text{cell}([\mathbf{i}]_2)$, there exists a continuously differentiable curve with maximum curvature κ traversing the remainder of the tile if the initial tangent angle of this curve lies in the interval $[\underline{\alpha}_1(w), \bar{\alpha}_1(w)]$. All of the quantities involved here are expressed in the local coordinate axes system attached to the second cell of the tile, i.e. $\text{cell}([\mathbf{i}]_2)$.
2. For every point with coordinates $(0, w)$ on the cell boundary $\text{cell}([\mathbf{i}]_2) \cap \text{cell}([\mathbf{i}]_3)$, there exists a continuously differentiable curve with maximum curvature κ traversing the remainder of the tile if the initial tangent angle of this curve lies in the interval $[\underline{\alpha}_2(w), \bar{\alpha}_2(w)]$. All of the quantities involved here are expressed in the local coordinate axes system attached to the third cell of the tile, i.e. $\text{cell}([\mathbf{i}]_3)$.

Note that these functions represent the backward reachable sets $\mathcal{Q}(\cdot)$ defined in Section 2.2.1.

The forward reachability sets $\mathcal{R}(\xi_s)$ are relatively easier to compute [114]. In the present context, we leverage the assumption (stated in Section 2.1.1) $\rho v_{\min} > 3$ and obtain conservative approximations to these sets in the form of regions described by $[\underline{w}, \bar{w}] \times [\underline{\beta}, \bar{\beta}] \times [\underline{v}, \bar{v}]$. The derivation and exact expressions for the quantities $\underline{w}, \bar{w}, \underline{\beta}, \bar{\beta}, \underline{v}, \bar{v}$, all of which depend on ξ_s , are lengthy and cumbersome, and are therefore omitted. For the reader's convenience, MATLAB[®] code implementing these computations is made available at <http://users.wpi.edu/~rvcowlagi/software.html>.

The preprocessing stage in the proposed route-planning algorithm involves the computation of these forward- and backward reachable set for each element of the tile library for each value of H . Furthermore, recall that the curvature constraints of interest for the vehicle model considered in this paper are speed-dependent. Therefore, these computations are

in fact performed for several different values of the curvature bound. The details of these preprocessing computations with precise index notations are discussed next.

Note that the results of computations of these forward and backward reachable sets constitute large volumes of numerical data, which is not advisable for storage and online lookup (even if the computations themselves are preprocessed offline). To resolve this issue, we store only an “abstraction” of the relations between these reachable sets in the form of sparse connectivity matrices, as explained next.

Regions in \mathfrak{S} and Connectivity

Consider an edge $(\mathbf{i}, \mathbf{j}) \in E_H$, and the associated tile. As previously noted, we consider copies \mathfrak{S}_1 and \mathfrak{S}_2 of the parallelepiped $[0, 1] \times [-\frac{\pi}{2}, \frac{\pi}{2}] \times [v_{\min}, v_{\max}]$ associated with, respectively, the second and third cells in the tile. Each of \mathfrak{S}_1 and \mathfrak{S}_2 are partitioned into uniformly sized regions, i.e. each region in this partition is itself a parallelepiped of dimensions $\Delta_w := \frac{1}{N^w}$, $\Delta_\psi := \frac{1}{N^\psi}$, and $\Delta_v := \frac{1}{N^v}$, where $N^w, N^\psi, N^v \in \mathbb{Z}_{\geq 0}$ are prespecified. We denote these regions by $\sigma_1[m], \sigma_2[m]$, where $m = 0, 1, \dots, (N^w N^\psi N^v - 1)$, and the subscripts 1 and 2 indicate to which of the parallelepipeds \mathfrak{S}_1 and \mathfrak{S}_2 the regions belong. Each region $\sigma_n[m]$, $n \in \{1, 2\}$ is defined by the Cartesian product $[\underline{\sigma}_n^w[m], \bar{\sigma}_n^w[m]] \times [\underline{\sigma}_n^\psi[m], \bar{\sigma}_n^\psi[m]] \times [\underline{\sigma}_n^v[m], \bar{\sigma}_n^v[m]]$, where

$$\underline{\sigma}_n^w[m] := \text{mod}(\text{mod}(m, N^v N^\psi), N^\psi) \Delta_w, \quad \bar{\sigma}_n^w[m] := \underline{\sigma}_n^w[m] + \Delta_w, \quad (2.16)$$

$$\underline{\sigma}_n^\psi[m] := \left\lfloor \frac{\text{mod}(m, N^w N^\psi)}{N^\psi} \right\rfloor \Delta_\psi, \quad \bar{\sigma}_n^\psi[m] := \underline{\sigma}_n^\psi[m] + \Delta_\psi, \quad (2.17)$$

$$\underline{\sigma}_n^v[m] := \left\lfloor \frac{m}{N^\psi N^w} \right\rfloor \Delta_v, \quad \bar{\sigma}_n^v[m] := \underline{\sigma}_n^v[m] + \Delta_v. \quad (2.18)$$

We associate with each tile a sparse square connectivity matrix C with $N^w N^\psi N^v$ rows. The preprocessing stage of the proposed route-planning algorithm then consists of the following computations to be performed for each tile for each value of H :

1. Perform curvature-bounded traversal analysis [112] with N^v different curvature bounds, and for each determine numerical approximations of the four functions discussed in Section 2.4. Denote these functions by $\underline{\alpha}_1^\ell, \bar{\alpha}_1^\ell, \underline{\alpha}_2^\ell, \bar{\alpha}_2^\ell$, where $\ell = 0, 1, \dots, (N^v - 1)$. The different curvature bounds to be used for this analysis are the following:

$$\kappa_{\max, \ell} := ((\bar{\sigma}_n^v[m])^2 \rho)^{-1}, \quad m = 0, 1, \dots, (N^w N^\psi N^v - 1). \quad (2.19)$$

Note that although the index m takes values in a larger range compared to the index ℓ , by (2.18), the number of unique values of the right hand side of (2.19) is N^v . The expression (2.19) for curvature bounds arises from the previously discussed expression (2.15). Here, since the traversal analysis is performed for a range of speeds, the dependence on tangential acceleration can be ignored in favor of conservative approximations to the speed as necessary.

2. Determine an index set $\mathbf{m}_2 \subseteq \{0, 1, \dots, (N^w N^\psi N^v - 1)\}$ such that

$$\sigma_2[m] \cap \left(\bigcup_{w \in [0,1]} [\underline{\alpha}_2^\ell(w), \bar{\alpha}_2^\ell(w)] \right) \neq \emptyset, \quad \text{for each } m \in \mathbf{m}_2, \text{ where } \ell := \left\lfloor \frac{m}{N^\psi N^w} \right\rfloor + 1.$$

Informally, the index set \mathbf{m}_2 indicates the regions $\sigma_2[m]$ that have a nonempty intersection with the backward reachable set $\mathcal{Q}(\mathbf{j})$.

3. Determine an index set $\mathbf{m}_1 \subseteq \{0, 1, \dots, (N^w N^\psi N^v - 1)\}$ such that

$$\sigma_1[m] \subseteq \bigcup_{w \in [0,1]} [\underline{\alpha}_1^\ell(w), \bar{\alpha}_1^\ell(w)], \quad \text{for each } m \in \mathbf{m}_1, \text{ where } \ell := \left\lfloor \frac{m}{N^\psi N^w} \right\rfloor + 1.$$

Informally, the index set \mathbf{m}_1 indicates the regions $\sigma_1[m]$ that are included in the backward reachable set $\mathcal{Q}(\mathbf{i})$.

4. For each $m \in \mathbf{m}_1$,

- (a) Determine $\underline{w}_m, \bar{w}_m, \underline{\beta}_m, \bar{\beta}_m, \underline{v}_m, \bar{v}_m$, (explicit expressions omitted due to length; MATLAB[®] code available at <http://users.wpi.edu/~rvcowlagi/software.html>).

(b) Determine an index set $\mathbf{m}_2^r \subseteq \mathbf{m}_2$ such that

$$\sigma_2[p] \cap \left([\underline{w}_m, \bar{w}_m] \times [\underline{\beta}_m, \bar{\beta}_m] \times [\underline{v}_m, \bar{v}_m] \right) \neq \emptyset, \quad \text{for each } p \in \mathbf{m}_2^r. \quad (2.20)$$

(c) Assign $C_{mp} = 1$, where C_{mp} is the element in the m^{th} row and p^{th} column in C . This step records connectivity between the m^{th} region in \mathfrak{S}_1 with the p^{th} region in \mathfrak{S}_2 .

2.5 Illustrative Numerical Simulation Results

Figures 2.8–2.10 illustrate applications of the proposed route guidance algorithm. In each of these results, a workspace partition with uniformly-sized square cells is considered. For clarity, the indices assigned to all cells are not shown; instead, different colors are used to indicate associations with atomic propositions, as in (2.4). In what follows, we use the notation of Section 2.1. All of these illustrative examples assume a constant unit speed of traversal, i.e. $u_1 = 0$. For preprocessing the connectivity matrices, the values $N^w = N^\psi = 100$ are used. Also, all of these illustrative examples are set up in a cell decomposition consisting of $N^C = 144$ cells.

For the result shown in Fig. 2.8(a), the number of atomic propositions is $N^R = 3$. In each of the following examples, the atomic proposition λ_1 is associated with all cells, including those indicated in white color, to establish workspace limits. The cells associated with atomic propositions λ_2 , λ_3 , and λ_4 are indicated in Fig. 2.8(a) by gray (obstacles), red, and yellow colors respectively. The dark green cell in the lower left corner is the cell containing $x(\xi_0)$. The sequence of numbered cells with bold outlines in Figure 2.8(a) indicate the route obtained as a result of searching the product transition system described in Section 2.3, for the LTL $_{-X}$ formula $\phi_1 := \square \lambda_1 \wedge \square \neg \lambda_2 \wedge \diamond \lambda_3$. This specification is read “*always* λ_1 , *never* λ_2 , and *eventually* λ_3 ”. Informally, this specification requires the vehicle to avoid the gray-colored regions and visit the red-colored region, and is equivalent to the standard

2.5. ILLUSTRATIVE NUMERICAL SIMULATION RESULTS

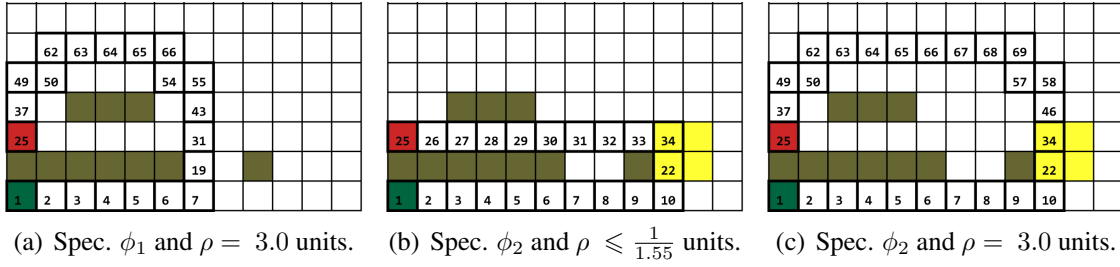


Figure 2.8: Application of the proposed approach: illustration of the effects on the resultant path of changes in the control input constraint. The numerical values ρ are in dimensionless distance units, where 1 unit is equal to the side of a cell in the uniform decomposition.

motion-planning problem of reaching a destination while avoiding obstacles. For the result indicated in Figure 2.8(a), the control input constraint parameter ρ is set to $\rho = 3$ units. Here, the unit of measurement is the size of a cell. The numbers indicated within cells in the resultant route are the indices assigned to those cells.

The sequences of numbered cells with bold outlines in Figs. 2.8(b) and 2.8(c) indicate the resultant route for satisfying the LTL_{-X} specification $\phi_2 := \phi_1 \wedge \diamond \lambda_4$. Informally, this specification requires the vehicle to avoid the gray-colored regions, and visit the red- and yellow-colored regions both. The different results in Figs. 2.8(b) and 2.8(c) are a consequence of changing the parameter ρ . In particular, for the result indicated in Fig. 2.8(b), $\rho \leq \frac{1}{1.55}$ units, whereas for the result indicated in Fig. 2.8(c) $\rho = 3$ units. Notice that the *same* LTL_{-X} specification is satisfied by two markedly different paths due to the different input constraints. The result shown in Fig. 2.8(b) is computed by ignoring the vehicle model altogether during route-planning and relies on the previously known result [115] that if a channel is sufficiently “wide,” (specifically, at least 1.55 times the minimum radius of turn) then a curvature-bounded curve is guaranteed to traverse it from any initial condition.

For the result shown in Fig. 2.9, the number of atomic propositions is $N^R = 3$. The cells associated with atomic propositions λ_2 and λ_3 are indicated in Fig. 2.9 by red and yellow colors. Informally, this specification requires the vehicle visit the red- and yellow-colored regions both. However, the order of visit is not specified, and there is no preference for *which* of the two yellow- or two red-colored regions is to be visited. The dark green

2.5. ILLUSTRATIVE NUMERICAL SIMULATION RESULTS

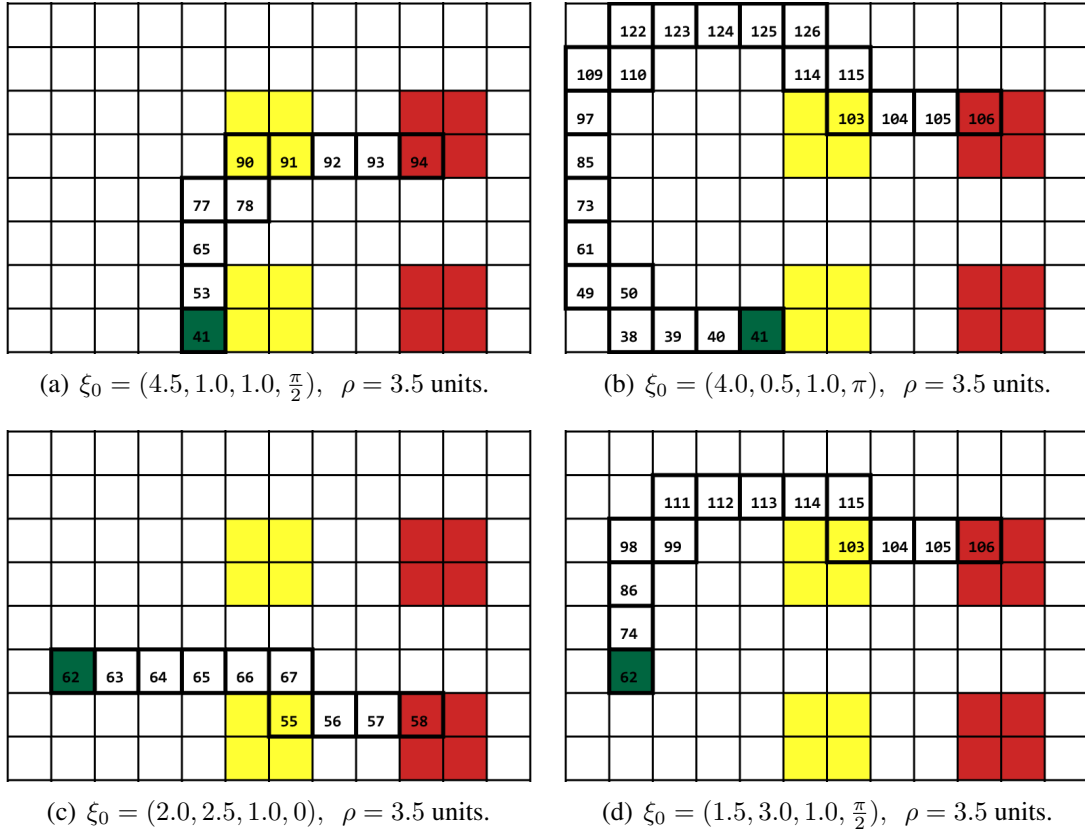


Figure 2.9: Application of the proposed approach: illustration of the effects on the resultant path of changes in the initial state. Here, the LTL_X specification is ϕ_3 (described in text). The numerical values of position coordinates and of ρ are in dimensionless distance units, where 1 unit is equal to the side of a cell in the uniform decomposition.

cell near the center is the cell containing $x(\xi_0)$. The sequence of numbered cells with bold outlines in each of Figs. 2.9(a)–2.9(d) indicate the route obtained as a result of searching the product transition system, for the LTL_X formula $\phi_3 := \square\lambda_1 \wedge \diamond\lambda_2 \wedge \diamond\lambda_3$. Notice that the regions associated with the propositions λ_2 and λ_3 (the red- and yellow-colored regions, respectively) are unions of disconnected subregions. The different resultant paths in each of Figs. 2.9(a)–2.9(d) are due to different initial states, as indicated in the captions.

Two additional results are shown in Fig. 2.10 to illustrate the impact of nonholonomic constraints on the manner in which the given LTL specifications are satisfied, and of the effectiveness of the proposed route-planning algorithm in handling these situations. For the result shown in Fig. 2.10(a), the number of atomic propositions is $N^R = 4$, with atomic

2.5. ILLUSTRATIVE NUMERICAL SIMULATION RESULTS

proposition λ_2 associated with gray regions, and propositions λ_3, λ_4 associated with the red- and yellow-colored regions respectively. The dark green-colored cell at the bottom left is the cell containing $x(\xi_0)$. The sequence of numbered cells with bold outlines in Fig. 2.10(a) indicates the route resulting from the proposed algorithm for satisfying the LTL_{-X} specification $\phi_4 := \phi_2 \wedge (\neg\lambda_4 \triangleright \lambda_3)$, which requires not only that the regions associated with λ_3 and λ_4 be both visited, but also that the region associated with λ_3 be visited first. The result shown in Fig. 2.10 is of significance for intelligence, surveillance, and reconnaissance UAV operations, where mission requirements may dictate the order of visit to different regions.

For the result shown in Fig. 2.10(b), the number of atomic propositions is $N^R = 5$, with atomic proposition λ_2 , associated with gray regions, the propositions λ_3 and λ_4 associated with the red- and yellow-colored regions, respectively, and the proposition associated with the two green-colored regions. The green-colored cell at the bottom left is the cell containing $x(\xi_0)$. The LTL_{-X} specification $\phi_5 := \phi_4 \wedge \diamond\Box\lambda_5$ is considered.

Informally, this specification requires that the gray-colored regions be avoided, that the red-colored region be visited, followed by the yellow-colored region, and that the route terminate in either one of the green-colored regions. This specification is simple, yet of immense practical significance: the sub-formula $(\diamond\Box\lambda_5)$ codifies the common – and necessary – mission requirement of “*return UAV to base*,” where the “base” location need not be unique (e.g. there may be multiple landing strips available).

The sequence of numbered cells with bold outlines indicates the route determined by the proposed route-planning algorithm. This route includes a cycle, and is defined by the sequence of cells

$$(1, \dots, 10, 22, 23, 35, \dots, 95, 94, \dots, 33, 34, 46, 47, 59, 60, 72 \dots, 144).$$

Notice that this result is significantly impacted by the control input constraints. Specifically, the route for a holonomic vehicle in this case may have simply involved returning to the green-colored cell (indicated with index 1) in the bottom left after visiting the red- and

2.5. ILLUSTRATIVE NUMERICAL SIMULATION RESULTS

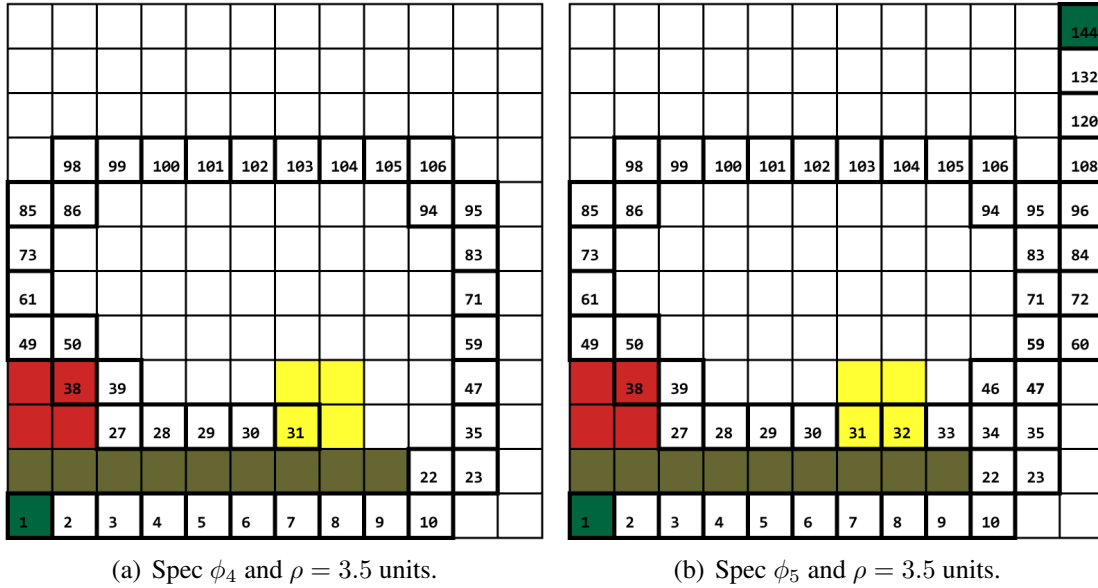


Figure 2.10: Additional examples that illustrate the significant impact of nonholonomic motion constraints on the manner in which the given LTL specifications are satisfied (details in text).

yellow-colored regions. However, such a route is infeasible with the control input constraint indicated in the figure caption. Furthermore, such a planned route (possibly computed by a naive route-planning algorithm) would have been a particularly adverse plan, considering the UAV’s kinematic constraints. This example therefore indicates the benefit offered by the proposed approach in finding routes that not only satisfy given LTL specifications, but are also compatible with the vehicle model and its constraints.

Figures 2.8–2.10 represent the results of executing the proposed route-planning algorithm with parameter $H = 5$. The algorithm is implemented in the MATLAB[®] (version R2014b), and executed on a desktop computer with an Intel[®] Core[™] i7 2.80 GHz processor, 16 GB RAM, and the Windows[®] 7 Enterprise 64-bit operating system. Representative computation times involved in the proposed approach are provided in Tables 2.1 and 2.2.

The numbers of vertices in lifted graphs \mathcal{G}_H , for several different values of the parameter H , are provided in Table 2.1. The computation time τ_{tplg} in the third row of Table 2.1 is the time required to record the topology of \mathcal{G}_H , i.e. to find and store in memory a list of the vertices and edges in the lifted graph. This may be considered a part of the offline

2.5. ILLUSTRATIVE NUMERICAL SIMULATION RESULTS

H	0	1	2	3	4	5	6	7
$ V_H $	144	528	1,448	3,072	6,832	15,032	33,088	71,200
τ_{tplg} (s)	—	0.0238	0.0895	0.2633	0.9265	3.347	12.95	53.46

Table 2.1: Number of vertices in the lifted graph, for various values of the parameter H , and the time required to record the graph topology.

preprocessing stage, because the edge transition costs in \mathcal{G}_H are not assigned at this stage.

The computation times for setting up and executing the proposed route-planning algorithm for the examples shown in Figs. 2.8–2.10 are provided in Table 2.2. Here, τ_{setup} is the total time required to record the lifted graph topology, to construct the Büchi automaton (using the LTL2BA algorithm [91], and its MATLAB[®] adaptation [?]), and to enlist and store transitions in the product transition system $\mathcal{T}_{\phi,H}$. The time τ_{search}^0 is the time required to search the product transition system using Dijkstra’s algorithm to find a route. The times τ_{search}^1 and τ_{search}^2 are the times required to search the product transition system using an A*-like informed search algorithm with a search heuristic. A full discussion of these search heuristics is beyond the scope of this paper, and the τ_{search}^1 and τ_{search}^2 are provided to indicate potential improvements in the execution time of the proposed algorithm. Briefly, these search heuristics are based on a preliminary search in the product transition system $\mathcal{T}_{\phi,0}$, which executes quickly. The data reported in Tables 2.1 and 2.2 are averages over three simulation trials for each specification, each value of the parameter H , and each value of the control constraint ρ for specifications ϕ_1 and ϕ_2 .

2.5.1 Discussion

The proposed route-planning algorithm offers several advantages compared to the state-of-the-art. First, a significant portion of the computations involved can be preprocessed (as discussed in Section 2.4), which reduces the online computational burden. For the numerical simulation results shown in Figures 2.8–2.10, the time to execute the preprocessing step with $N^w = N^\psi = 100$, $N^v = 1$, $H = 1, \dots, 7$, and $\rho = 3.5$, on the aforementioned

2.5. ILLUSTRATIVE NUMERICAL SIMULATION RESULTS

Specification \rightarrow	ϕ_1			ϕ_2			ϕ_3	ϕ_4	ϕ_5
$\rho \rightarrow$	3.0	4.0	5.0	3.0	4.0	5.0	3.5	3.5	3.5
H = 3, $ \mathcal{T}_{\phi,H} $	6,146			12,292			12,292	9,219	18,438
$ \delta_{\phi}H $	11,110			22,892			32,308	17,771	17,963
τ_{setup} (s)	2.429			5.040			4.917	—	—
τ_{search}^0 (s)	6.391	3.705	—	18.82	6.992	2.258	0.9605	—	—
τ_{search}^1 (s)	1.049	2.036	—	11.47	6.076	2.251	0.6679	—	—
H = 4, $ \mathcal{T}_{\phi,H} $	13,666			27,332			27,332	20,499	40,998
$ \delta_{\phi}H $	23,799			49,114			73,608	38,496	38,904
τ_{setup} (s)	6.545			13.73			13.73	—	—
τ_{search}^0 (s)	14.88	6.083	1.985	42.09	13.95	3.814	1.398	—	—
τ_{search}^1 (s)	2.272	2.758	1.995	24.88	10.02	3.563	0.8846	—	—
H = 5, $ \mathcal{T}_{\phi,H} $	30,066			60,132			60,132	45,099	90,198
$ \delta_{\phi}H $	51,114			105,660			167,582	83,563	84,475
τ_{setup} (s)	19.84			39.12			37.64	35.68	71.56
τ_{search}^0 (s)	22.42	12.90	6.205	69.69	34.29	9.443	1.436	38.08	66.78
τ_{search}^1 (s)	2.421	3.462	6.028	37.20	21.84	9.132	0.9781	35.90	84.27
τ_{search}^2 (s)	Experiment not performed.							31.65	49.26

Table 2.2: Execution times for the numerical simulation examples discussed in Section 5.4. The blank entries for τ_{search}^0 and τ_{search}^1 for $H = 3, 4$ indicate that the search returned failure after a finite number of iterations, and no route was found.

2.5. ILLUSTRATIVE NUMERICAL SIMULATION RESULTS

desktop computer, is approximately 65 minutes. Connectivity matrices for each of a total of 254 tiles (unique up to rigid transformations) are stored. The total number of non-zero elements in all of these matrices is approximately 64.21×10^6 , which requires memory space of approximately 32 MB using the sparse matrix data structure provided by MATLAB[®].

Whereas the execution times reported in Table 2.2 are of the order of seconds or tens of seconds, we claim that significant portions of these execution times are merely due to the computational overhead introduced by MATLAB[®]. This claim is corroborated by the fact that the sizes of the product transition systems, in terms of numbers of states, indicated in Table 2.2 are relatively small in comparison to the typical graph sizes (of the order of 10^7 vertices) that can be searched by high-performance software libraries within a few tens of seconds [116, 117]. The main difficulty in the proposed approach is that the transition costs in the product transition system depend on computationally intensive reachability calculations. However, the offline preprocessing stage as discussed in Section 2.4 enables the determination of these transition costs via a simple lookup table. Therefore, it is possible to reduce these execution times by several orders of magnitude by appropriately implementing the proposed algorithm with a low-level programming language such as C++, and by utilizing high-performance open-source software libraries. Appropriate search heuristics can also significantly reduce the execution times, and enable real-time implementations of the proposed algorithm.

Second, as previously noted for the example shown in Fig. 2.9, the proposed approach allows for the association of disconnected regions in the workspace with a single atomic proposition. Consequently, the total number of propositions can be reduced, thereby reducing the size of the product transition system. This ability is not available in other approaches reported in the literature, e.g. [87, 96], which depend on partitioning the vehicle's *state space* instead of its workspace (output space).

Third, the proposed approach does not use up control authority for the sake of generating a discrete abstraction of the vehicle dynamical model, as is often done in the literature [4, 6, 87, 96, 118]. This approach leaves room to accommodate any independent trajec-

2.5. ILLUSTRATIVE NUMERICAL SIMULATION RESULTS

tory optimization algorithm to be employed for generating reference trajectories for tracking the planned route. The trajectory optimization problem is subjected to the constraint that the workspace projection of the resulting must remain within the channel associated with the planned route (namely, the channels indicated by numbered cells with bold outlines in Figs. 6–8).

Chapter 3

Motion-planning for Multiple Nonholonomic Vehicles

In this chapter we propose a centralized motion-planning algorithm for a team of robotic vehicles subject to nonholonomic kinematic constraints and global LTL specifications. The problem formulation still relies on workspace cell decompositions, where certain regions of interest in the robots' shared workspace are defined. The proposed algorithm involves two graphs: first, the topological graph \mathcal{G} arising from the workspace cell decomposition, and second, a graph \mathcal{G}^R arising from vertex aggregation on \mathcal{G} , such that each region of interest is a vertex \mathcal{G}^R . We still apply the *method of lifted graphs* to determine feasibility of edge transitions in \mathcal{G} and \mathcal{G}^R . Next, a product transition system is constructed from this team transition system and the Büchi automaton associated with the global LTL specifications. Runs of this product transition system can be uniquely projected to paths (for each vehicle) that are compatible with the nonholonomic constraints and also ensure that the global LTL specifications are satisfied.

3.1 Problem Formulation

We use the same vehicle model as we used before for each robotic vehicle in a team. And we will introduce some new preliminary ideas involved in this problem next.

3.1.1 Regions of Interest Graph

In the workspace cell decomposition as we defined before, special cells we called *base locations* and *regions of interest (ROI)* are prespecified. The number of base locations is equal to the number N^V of robotic vehicles in the team, and each robot is assumed to start from one of these locations. The vertices in V associated with these cells are denoted by $v_{B,1}, \dots, v_{B,N^V}$. The number of ROIs is N^R . Each ROI is a connected union of cells, i.e. the k^{th} ROI is $\cup_{i \in \varsigma_k} c[i]$, where $\varsigma_k \subseteq \{1, \dots, N^C\}$, $k = 1, \dots, N^R$ are prespecified. The associated vertices are denoted $v_{R,\ell k}$, $\ell \in \varsigma_k$.

The motion of the entire team of vehicles can be modeled by N^V copies of \mathcal{G}_H . Specifically, the idea is to construct a *team transition system* [84] through a Cartesian product-like operation with N^V copies of \mathcal{G}_H . Whereas this approach is viable, it is wasteful when the number of ROI N^R is small compared to the number of cells N^C . Instead, we consider an undirected graph – henceforth referred to as the *ROI graph* – $\mathcal{G}^R = (V^R, E^R)$, which is constructed by aggregating vertices in \mathcal{G} . To this end, we introduce a map $ROI : 2^V \rightarrow V^R$ that uniquely associates each vertex in V^R with either a ROI or a robot base, i.e. the total number of vertices in V^R is $N^R + N^V$. Precisely:

$$V^R := \bigcup_{k=1, \dots, N^R} ROI(\{v_{R,\ell k}\}_{\ell \in \varsigma_k}) \cup \bigcup_{k=1, \dots, N^V} ROI(v_{B,k}).$$

The edge set E^R is defined to be complete, i.e. each vertex in V^R is adjacent to every other vertex in V^R . Informally, paths in \mathcal{G}^R “abstractly” denote vehicle routes to complete the specified task (LTL formula). In a minor abuse of notation we denote by ROI^{-1} the set association of every element $q \in V^R$ with vertices in V , e.g., $ROI^{-1}(q) = \{v_{R,\ell k}\}_{\ell \in \varsigma_k}$.

The finite state model of the vehicle team in the proposed approach is intricately linked with the motion-planning algorithm. Informally, this finite state model is obtained by lifting \mathcal{G}^R to construct \mathcal{G}_M^R for $M \geq 0$ followed by a product-like operation with N^V copies of \mathcal{G}_M^R . Transition costs in \mathcal{G}_M^R are assigned by computing optimal paths in \mathcal{G}_H . Figure 3.1 illustrates

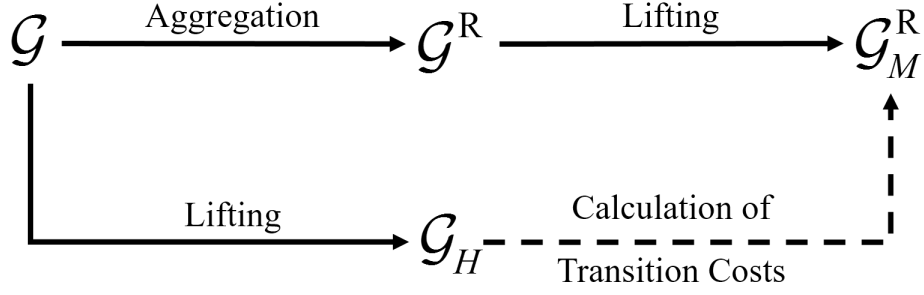


Figure 3.1: Conceptual relationship between the various graphs involved.

the relationship between the various graphs discussed here.

3.1.2 Global LTL_{-X} specifications

Consider paths $\mathbf{v}_n \in \mathcal{L}_{\mathcal{G}}$, $n = 1, \dots, N^V$, associated with the motion of each vehicle in the team. Each of these paths defines a word $\bar{\omega}_n(\mathbf{v}_n) = (\omega_{0,n}, \omega_{1,n}, \dots)$, which we “concatenate” to define a word for the entire team as follows

$$\bar{\omega}(\pi) := (\omega_{0,1}, \omega_{0,2}, \dots, \omega_{1,1}, \omega_{1,2}, \dots), \quad (3.1)$$

where $\pi := (\mathbf{v}_1, \dots, \mathbf{v}_{N^V})$. Here, the rule of “concatenation” is that ω_{ℓ_1, n_1} appears before ω_{ℓ_2, n_2} in $\bar{\omega}(\pi)$ if $\ell_1 < \ell_2$ or else if $\ell_1 = \ell_2$ and $n_1 < n_2$, for $\ell_1, \ell_2 \in \mathbb{Z}_+$ and $n_1, n_2 \in \{1, \dots, N^V\}$. The n -tuple of paths π is said to collectively satisfy a formula ϕ if $\bar{\omega}(\pi)$ satisfies ϕ . The main problem of interest is then formulated as follows.

Problem 2. Given a LTL_{-X} formula ϕ over Λ , and vehicle initial conditions $\xi_{0,n} \in \mathcal{D}$, for $n = 1, \dots, N^V$, determine

$$\mathcal{L}_{\Gamma\phi} \subseteq \mathcal{L}_{\Gamma}(\xi_{0,1}) \times \dots \times \mathcal{L}_{\Gamma}(\xi_{0,N^V})$$

such that every N^V -tuple of paths in $\mathcal{L}_{\Gamma\phi}$ collectively satisfies the formula ϕ . \square

An instance of Problem 3 with two vehicles is illustrated in Fig. 3.2, where four ROIs

19	20	21	22	23	24	25	26	27
10	11	12	13	14	15	16	17	18
1	2	3	4	5	6	7	8	9

Figure 3.2: An instance of Problem 1. Here, ROIs λ_1 and λ_2 are indicated in red (cells 19 and 27, respectively), and ROIs λ_3 and λ_4 are indicated in gray (cells 10–12 and 16–18, respectively).

(gray and yellow) and two base locations (green) are indicated. The given LTL specification is $(\diamond\lambda_1) \wedge (\diamond\lambda_2) \wedge (\square\neg\lambda_3) \wedge (\square\neg\lambda_4)$. This formula specifies that the yellow-colored ROIs must be eventually visited (no preference for the order of visit), and that the gray-colored ROIs must be always avoided. Intuitively, the expected motion plan will involve a vehicle from base $B1$ to visit ROI λ_1 and the second vehicle from base $B2$ to visit ROI λ_2 . However, due to nonholonomic constraints, the sharp turns required to execute this plan may be infeasible, and an alternative plan is required.

3.2 Multi-Vehicle Motion-planning

For $H, M \geq 1$, consider \mathcal{G}_M^R . According to the definition of lifting in Section 2.2, each edge transition in \mathcal{G}_M^R corresponds to a sequence of $M + 2$ locations (either ROIs or base locations) in \mathcal{G} . Accordingly, we can assign a transition cost to this edge in \mathcal{G}_M^R based on a vehicle's cost of traversal between these $M + 2$ locations.

To this end, the *team state* $\mu = (\mathbf{q}_1, \dots, \mathbf{q}_{N^V}) \in (V_M^R)^{N^V}$ is defined as the N^V -tuple of vertices in V_M^R indicating the location of each vehicle. Specifically, the team state μ indicates that the k^{th} vehicle is at the location associated with $[\mathbf{q}_k]_1 \in V^R$. We denote by Q_M the set of all team states. A *team state transition* is the relation $\delta_R \subseteq (V_M^R)^{2N^V}$ defined by

$$(\mu, \gamma) \in \delta_R \text{ iff } (\mathbf{q}_k, \mathbf{r}_k) \in E_M^R, \text{ for each } k = 1, \dots, N^V,$$

3.2. MULTI-VEHICLE MOTION-PLANNING

where $\mu = (\mathbf{q}_1, \dots, \mathbf{q}_{N^V})$ and $\gamma = (\mathbf{r}_1, \dots, \mathbf{r}_{N^V})$. Note that each $(\mu, \gamma) \in \delta_R$ is associated with N^V sequences $\{([\mathbf{q}_k]_1, \dots, [\mathbf{q}_k]_{M+1}, [\mathbf{r}_k]_{M+2})\}$ of vertices of V^R , where $k = 1, \dots, N^V$. Let $\mathbf{v}_k^*(\mu, \gamma) \in \mathcal{L}_G$ denote a path with minimal H -cost that passes through all of the vertices $ROI^{-1}([\mathbf{q}_k]_1), \dots, ROI^{-1}([\mathbf{q}_k]_{M+1}), ROI^{-1}([\mathbf{r}_k]_{M+2})$ in that order. Then we define the cost of the transition (μ, γ) by $\sum_{k=1}^{N^V} \mathcal{J}_H(\mathbf{v}_k^*(\mu, \gamma))$.

The computation of these transition costs is time-consuming, but these computations can be preprocessed offline, and therefore do not constitute a computational burden on the proposed motion-planning algorithm.

Assuming that the initial location of the k^{th} vehicle is the k^{th} base location, we can define a set $Q_{0,M} \subset (V_M^R)^{N^V}$ of initial team states as:

$$Q_{0,M} := \{\mu \in Q_M \mid [\mathbf{q}_k]_1 = ROI(v_{B,k}), \quad k = 1, \dots, N^V\}.$$

The triplet $\mathcal{Q}_M = (Q_M, Q_{0,M}, \delta_R)$ defines the *team state transition system*, which is a finite state model of the motion of the entire vehicle team.

Then, we define a *product transition system* $\mathcal{T}_{\phi,M} := (T, \delta_{\mathcal{T}_{\phi,M}})$ as the product of \mathcal{Q} and \mathcal{B}_ϕ as follows:

1. The set of states of $\mathcal{T}_{\phi,M}$ is $T := S \times Q_M$. For every state $\theta \in T$, we denote by $\theta|_S$ and $\theta|_{Q_M}$, respectively, the projection of θ on S and Q_M .
2. The transition relation of $\mathcal{T}_{\phi,M}$ is $\delta_{\mathcal{T}_{\phi,M}} \subseteq T \times 2^\Lambda \times T$ defined as the set of all triplets $(\theta_k, \omega_k, \theta_\ell)$ such that

$$(\theta_k|_S, \omega_k, \theta_\ell|_S) \in \delta_{\mathcal{B}_\phi}, \quad (\theta_k|_{Q_M}, \theta_\ell|_{Q_M}) \in \delta_R, \quad (3.2)$$

$$\omega_k = \{\lambda_i \mid \theta_k|_{Q_M} \subseteq (\mathbf{q}_{R,1}, \dots, \mathbf{q}_{R,N^V})\}. \quad (3.3)$$

Here $\mathbf{q}_{R,n} \in V_M^R$ is a tuple that contains $ROI(\{v_{R,\ell_i}\}_{\ell \in \mathcal{S}_i})$.

A *run* of $\mathcal{T}_{\phi,M}$ is a sequence $\Theta = (\theta_0, \theta_1, \dots)$ such that $\theta_k \in T$ for each $k \in \mathbb{N}$, and

3.2. MULTI-VEHICLE MOTION-PLANNING

$(\theta_k, \omega_k, \theta_{k+1}) \in \delta_{\mathcal{T}_{\phi, M}}$, with ω_k as defined in (3.3). We denote by $\Theta|_S = (\theta_0|_S, \theta_1|_S, \dots)$ and $\Theta|_{Q_M} = (\theta_0|_{Q_M}, \theta_1|_{Q_M}, \dots)$, respectively, the projections of Θ on S and Q_M . Note that $\Theta|_{Q_M}$ corresponds to N^V paths in $\mathcal{L}_{\mathcal{G}}$, i.e. one path for each vehicle in the team. The k^{th} path $\pi_k(\Theta) \in \mathcal{L}_{\mathcal{G}}$ is defined by the concatenation

$$\pi_k(\Theta) := (\mathbf{v}_k^*(\theta_0|_{Q_M}, \theta_1|_{Q_M}), \mathbf{v}_k^*(\theta_1|_{Q_M}, \theta_2|_{Q_M}), \dots)$$

Similar to [84, 87], we restrict attention to runs of $\mathcal{T}_{\phi, M}$ of a “prefix-suffix” form $\Theta = (\Theta_p, \Theta_s, \Theta_s, \dots)$. Here, the “suffix” run $\Theta_s = (\theta_f, \dots, \theta_f)$, which is repeated infinitely often in Θ , is a finite sequence such that $\theta_f \in S_f \times Q_M$. The “prefix” run $\Theta_p = (\theta_0, \dots, \theta_P)$ is a finite sequence such that $\theta_0 \in S_0 \times Q_M$ and $(\theta_P, \omega_P, \theta_f) \in \delta_{\mathcal{T}_{\phi, M}}$.

Now we state the main result of this part of work as follows.

Theorem 2. *Let $\Theta = (\Theta_p, \Theta_s, \Theta_s, \dots)$ be a run of $\mathcal{T}_{\phi, M}$. If $\mathcal{J}_H(\pi_k(\Theta_p)) < \chi$ and $\mathcal{J}_H(\pi_k(\Theta_s)) < \chi$, for each $k = 1, \dots, N^V$, then*

$$(\pi_1(\Theta), \dots, \pi_{N^V}(\Theta)) \in \mathcal{L}_{\Gamma\Phi}.$$

The proposed algorithm is shown in pseudocode-form Fig. 3.3. A fixed value of $H \geq 1$, with higher values of H preferred. As a preprocessing step, we compute the optimal paths $\mathbf{v}_k^*(\mu, \gamma)$ for all transitions (μ, γ) in $\delta_{\mathcal{R}}$, for various values of M , and the H -costs of these paths are stored in a lookup table.

The main algorithm is initialized with $M = 0$, and an optimal run in the product transition system $\mathcal{T}_{\phi, M}$ is identified using, say, Dijkstra’s algorithm. This run is projected to paths for individual vehicles, and the H -costs of these paths are calculated. By Prop. 1, these paths are feasible for traversal by admissible vehicle trajectories iff their H -costs are less than χ . If all the paths are found to be feasible, then their total cost is recorded and a feasible solution to the overall problem is now available. The algorithm then increments M , either to determine a first feasible solution, or to reduce the cost of the last known feasible solution. This

Incremental Multi-Vehicle Motion-planning

```

1:  $M := 0$ , choose  $H \geq 1$ 
2: total_cost :=  $\chi N^V$ 
3: while true do
4:   Find an optimal run  $\Theta^*$  of  $\mathcal{T}_{\phi, M}$ 
5:   all_paths_feasible := true
6:   for  $k = 1, \dots, N^V$  do
7:     if  $\mathcal{J}_H(\pi_k(\Theta^*)) \geq \chi$  then
8:       all_paths_feasible := false
9:       break;
10:  if all_paths_feasible then
11:    total_cost :=  $\min\left(\text{total\_cost}, \sum_{k=1}^{N^V} \mathcal{J}_H(\pi_k(\Theta^*))\right)$ 
12:     $M := M + 1$ 

```

Figure 3.3: Pseudocode description of the proposed incremental algorithm for solving the multi robot task/path planning problem.

algorithm can be terminated either when the first feasible solution is found, or whenever a predetermined maximum execution time is reached.

3.3 Illustrative Numerical Simulation Results

Figure 3.4 illustrates the application of the proposed algorithm to the sample problem introduced in Fig. 3.2. The proposed motion-planning algorithm starts with $M = 0$, and preprocessed information with $H = 0$. The product transition system $\mathcal{T}_{\phi, 0}$ is searched. The resulting paths for each vehicle is shown in Fig. 3.4(a), which is a intuitively expected solution. Since $H = 0$, no information regarding the vehicle dynamical model is included.

The result of executing the proposed algorithm with $M = 0$ and $H = 3$ is shown in Fig. 3.4(b). Here, the minimum H -cost of any path between the base $B1$ and ROI λ_1 (and similarly, between base $B2$ and ROI λ_2) is found to be greater than χ . The solution illustrated

3.3. ILLUSTRATIVE NUMERICAL SIMULATION RESULTS

in Fig. 3.4(b) indicates that $\pi_2 = \emptyset$, and

$$\pi_1 = (1, 2, 3, 4, 13, 14, 15, 24, 25, 26, 27, 26, 25, 24, \\ 23, 22, 21, 20, 19).$$

These paths for the two vehicles satisfy the given specification of visiting ROIs 19 and 27, and avoiding the gray-colored ROIs. However, the overall path π_1 is still infeasible for traversal due to the sharp change in direction at ROI 27. The reason for this solution is that $M = 0$, and only single edge transitions in the ROI graph are encoded with H -costs of paths. In this case, with $H = 3$, the segments $(1, 2, 3, 4, 13, 14, 15, 24, 25, 26, 27)$ and $(27, 26, 25, 24, 23, 22, 21, 20, 19)$ of π_1 each have H -cost lower than χ , but their concatenation π_1 does not.

The algorithm finds this infeasibility of traversal in Line 9 of the proposed algorithm, and proceeds to search for another solution the the product transition system with $M = 1$. The result is illustrated in Fig. 3.4(c), which indicates the paths for the two vehicles as:

$$\pi_1 = (1, 2, 3, 4, 13, 14, 15, 24, 25, 26, 27), \\ \pi_2 = (9, 8, 7, 6, 15, 14, 13, 22, 21, 20, 19).$$

Both of these paths are feasible for traversal (i.e. the H -cost of each path is less than χ), and together the global specification is satisfied.

Figure 3.5 illustrates the result of application of the proposed algorithm to a 4-vehicle problem. The LTL specifications are similar to the previous problem.

Next, we address an issue that arises due to the simplifying assumption related to temporal synchronization, as discussed in Section 3.1. Consider the workspace shown in Fig. 3.6(a), where two base locations (green) and two ROIs to be visited (red) are indicated. If the order of visit is irrelevant (e.g. the LTL specification $\phi = (\diamond\lambda_1) \wedge (\diamond\lambda_2)$), then temporal synchronization is not important. In this case, the projection of a run of the product

3.3. ILLUSTRATIVE NUMERICAL SIMULATION RESULTS

system to Q_M , namely, $\Theta|_{Q_M}$ is

$$\Theta|_{Q_M} = \left((v_{B,1}, v_{B,2}), (v_{R,1}, v_{B,2}), (v_{R,1}, v_{R,2}) \right).$$

This result means that the paths (in the ROI graph) to be followed by the two vehicles are, respectively, $(v_{B,1}, v_{R,1})$ and $(v_{B,2}, v_{R,2})$ for the vehicles starting from base locations 1 and 2. These paths are illustrated in Fig. 3.6(b).

Now, consider the formula $\phi = \diamond(\lambda_2 \wedge (\diamond\lambda_1))$, which specifies that λ_2 be visited before λ_1 . In this case, the projection of a run of the product system to Q_M results in

$$\Theta|_{Q_M} = \left((v_{B,1}, v_{B,2}), (v_{B,1}, v_{R,2}), (v_{R,1}, v_{R,2}) \right).$$

Because we have excluded temporal synchronization, the paths for the individual vehicles in these two cases are the same. However, it is clear from the two expressions for $\Theta|_{Q_M}$ that temporal information is encoded within the ROI graph paths, and needs to additionally be encoded with time-stamps for temporal synchronization.

3.3. ILLUSTRATIVE NUMERICAL SIMULATION RESULTS

19	20	21	22	23	24	25	26	27
10	11	12	13	14	15	16	17	18
1	2	3	4	5	6	7	8	9

(a) Result of searching the product transition system with $M = 0, H = 0$.

19	20	21	22	23	24	25	26	27
10	11	12	13	14	15	16	17	18
1	2	3	4	5	6	7	8	9

(b) Result with $M = 0, H = 3$.

19	20	21	22	23	24	25	26	27
10	11	12	13	14	15	16	17	18
1	2	3	4	5	6	7	8	9

(c) Result with $M = 1, H = 3$.

Figure 3.4: Solutions to the motivating example in Fig. 3.2. The yellow- and blue-colored cells indicate, respectively, the paths planned for the first and second vehicle (additional details are in Section 5.4).

3.3. ILLUSTRATIVE NUMERICAL SIMULATION RESULTS

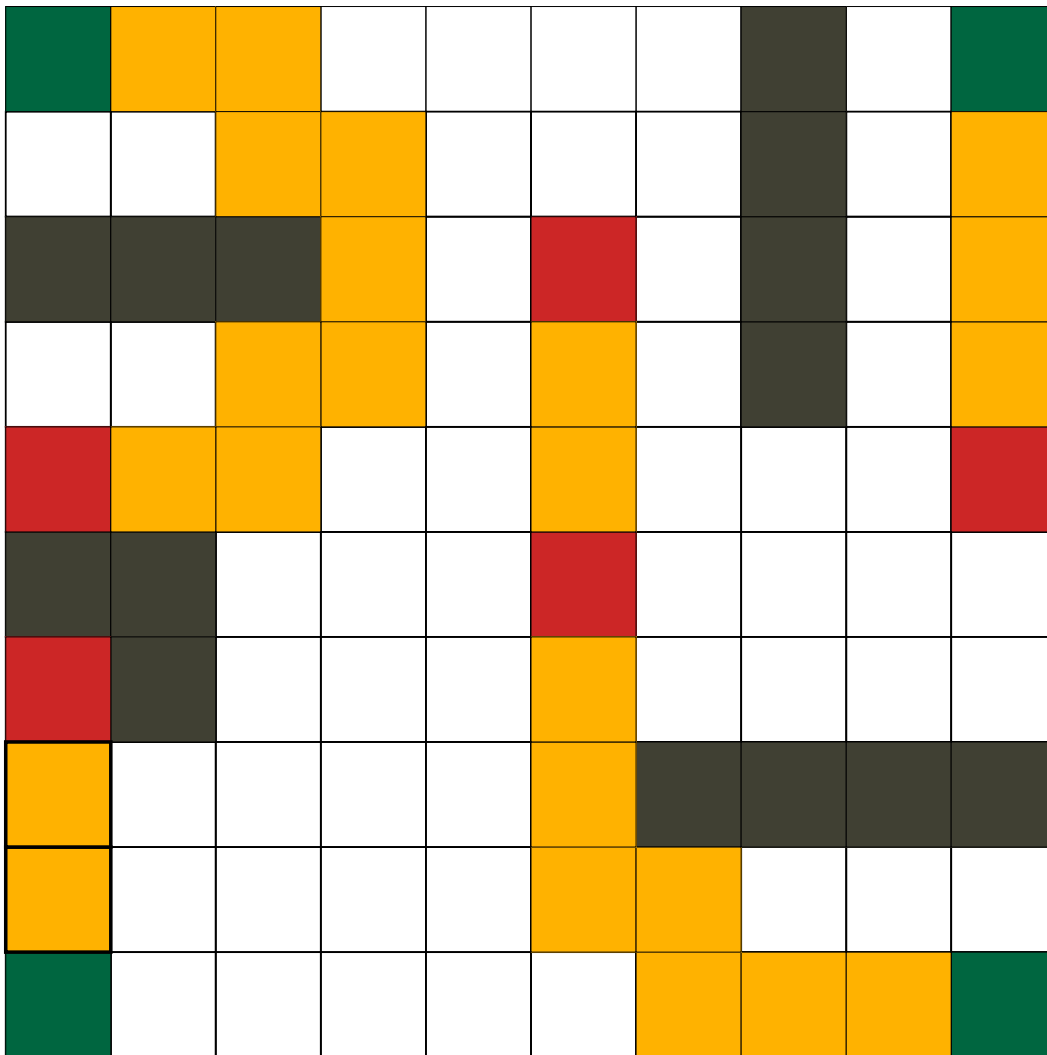
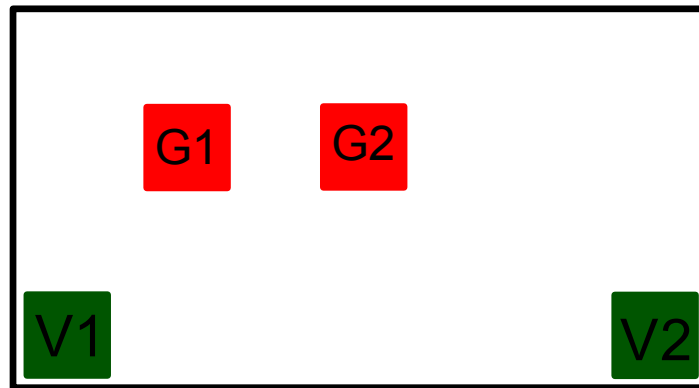
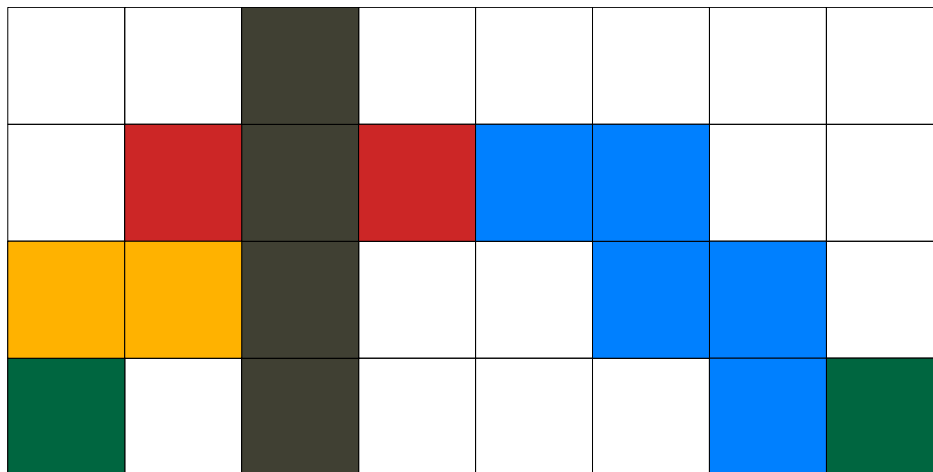


Figure 3.5: Illustration of motion-planning for 4 vehicles. The LTL specification is similar to that of the example in Fig. 3.2. Here, ROIs to be always avoided are indicated in gray, and ROIs to be eventually visited are indicated in red (order of visit is not relevant). The base locations are indicated in green, and the paths found for each vehicle are indicated in yellow.

3.3. ILLUSTRATIVE NUMERICAL SIMULATION RESULTS



(a) Conceptual illustration of workspace.



(b) Illustration of numerical simulation result.

Figure 3.6: Illustration of motion-planning when temporal synchronization is important. The paths of the two vehicle are indicated with yellow- and blue-colored cells.

Chapter 4

Incremental Motion-Planning

In this chapter we proposed an incremental algorithm in [119] for hierarchical motion-planning based on the previously developed H -cost motion-planning technique using the *method of lifted graphs*. The idea of incremental H -cost motion-planning retains the primary benefit of the original H -cost technique of finding an optimal high-level plan with the guarantee that there exists a state trajectory of Γ to execute this plan. Additionally, the proposed algorithm significantly mitigates the computational complexity of the original H -cost approach by introducing incremental computations. The primary motivation behind developing an incremental algorithm is as follows: the fundamental computational complexity in the H -cost technique (or any similar hierarchical motion-planning technique) cannot be defeated if the objective is solely to find an optimal motion plan. In the interests of real-time implementation, it is beneficial to develop a motion planner that returns a feasible solution at nearly any time during its search for an optimal motion plan.

A conceptual illustration of the incremental planning is shown in Fig. 4.1. Any path π in the graph \mathcal{G}_H is uniquely mapped to a path in the graph \mathcal{G}_{H+1} , and this mapping is bijective, for every $H \geq 0$. Let π_H^* be an optimal path $b(\pi)$ in \mathcal{G}_H , and let π_{H+1}^* be an optimal path in \mathcal{G}_{H+1} . So the idea is to construct π_{H+1}^* by making incremental changes to the path $b(\pi_H^*)$. The advantage of this process is that \mathcal{G}_H is a significantly smaller graph compared to \mathcal{G}_{H+1} , and therefore the search for the path π_H^* in \mathcal{G}_H will be faster than a direct search for π_{H+1}^* in \mathcal{G}_{H+1} . Furthermore, this process can be started with $H = 0$, and continued for the amount of time allotted during real-time computations.

The proposed incremental planning algorithm is motivated by the following property of the transition cost function of *lifted graph* \mathcal{G}_H .

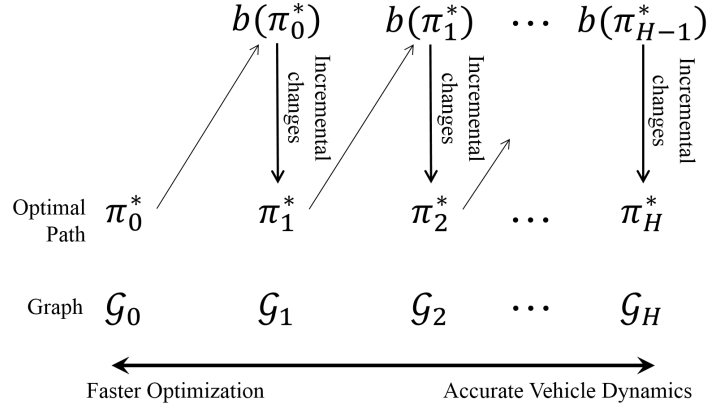


Figure 4.1: Illustration of proposed implementation of the H -cost framework with incremental planning

Proposition 2. Let $\pi = \{j_0, \dots, j_P\}$ be a path in the graph \mathcal{G} such that $\mathcal{J}_H(\pi) < \chi$. Let $I_k^H \in V_H$ be vertices defined by $I_k^H := (j_{k-H}, \dots, j_k)$, for $k \in \{H, \dots, P\}$. Then for each $\ell \in \{H+2, \dots, P\}$, either

$$\begin{aligned} \tilde{g}_{H+1}(I_{\ell-1}^{H+1}, I_{\ell}^{H+1}) &= \chi, \quad \text{or} \\ \tilde{g}_{H+1}(I_{\ell-1}^{H+1}, I_{\ell}^{H+1}) &\leq \tilde{g}_H(I_{\ell-2}^H, I_{\ell-1}^H) + \tilde{g}_H(I_{\ell-1}^H, I_{\ell}^H). \end{aligned}$$

If \tilde{g}_H , $H \geq 0$, are coarse transition cost functions, then the preceding inequality reduces to an equality:

$$\tilde{g}_{H+1}(I_{\ell-1}^{H+1}, I_{\ell}^{H+1}) = \tilde{g}_H(I_{\ell-2}^H, I_{\ell-1}^H) + \tilde{g}_H(I_{\ell-1}^H, I_{\ell}^H).$$

By definition, the sequence of vertices in V defined by the edge $(I_{\ell-1}^{H+1}, I_{\ell}^{H+1}) \in E_{H+1}$ is the same as that defined by the successive edges $(I_{\ell-2}^H, I_{\ell-1}^H), (I_{\ell-1}^H, I_{\ell}^H) \in E_H$. Also, for each $H \geq 0$ and $k \in \{H, \dots, P\}$, the vertex I_k^H belongs to the path $\pi^H = b_0^H(\pi)$. Proposition 2 states that either the edge $(I_{\ell-1}^{H+1}, I_{\ell}^{H+1})$ in graph \mathcal{G}_{H+1} is infeasible for traversal, or the transition cost of this edge is no greater than the sum of the transition costs of the successive edges $(I_{\ell-2}^H, I_{\ell-1}^H)$ and $(I_{\ell-1}^H, I_{\ell}^H)$ in graph \mathcal{G}_H , for each $\ell \in \{H+2, \dots, P\}$. Therefore, either $\mathcal{J}_{H+1}(\pi) \geq \chi$ or $\mathcal{J}_{H+1}(\pi) \leq \mathcal{J}_H(\pi)$.

Informally, this observation about $\mathcal{J}_{H+1}(\pi)$ means that a path π in \mathcal{G} , upon “further scrutiny” with higher values of the parameter H , will either be infeasible for traversal by trajectories of the vehicle dynamical system model Γ , or the cost of this traversal can potentially be reduced. Proposition 2 implies that it is desirable to use as large a value of the parameter H as possible. However, it is computationally impractical to do so, because the number of vertices in \mathcal{G}_H increases exponentially with H .

To address this issue, we propose an incremental algorithm to search for a \bar{H} -cost optimal path in \mathcal{G} . The main idea is to seed the search for this path with a path that is optimal in the graph \mathcal{G}_0 . The proposed algorithm uses a path repair procedure that replaces edges that are either infeasible or of high cost in lifted graphs. Starting with $H = 0$ (or any other small integer value of H that is convenient for computation of the initial path), the proposed algorithm incrementally progresses to higher values of H .

A precise description of the proposed incremental H -cost motion-planning algorithm is provided in Fig. 4.2. For simplicity of exposition, the algorithm in Fig. 4.2 considers coarse transition cost functions, and it can accommodate refined cost functions with no modifications in principle. In Fig. 4.2, the algorithm is initialized in Lines 1–2 by searching for a “seed” path in graph \mathcal{G}_0 with prespecified initial and goal vertices $j_S, j_G \in V$. The result of the algorithm at the n^{th} iteration, where $n \in \mathbb{Z}_+$, is a path π_n^0 in \mathcal{G} from the initial vertex j_S to the goal vertex j_G . Lines 4–19 describe the iterative process of the algorithm as it progresses through increasing values of H . Whenever H is incremented and an infeasible edge is detected in \mathcal{G}_H , the algorithm replaces this infeasible edge with a subpath with finite H -cost, as described in Lines 15–19. The path repair procedure in Line 16 computes this subpath from the vertex immediately before the infeasible edge to any of the vertices between the infeasible edge and the goal. In Lines 8–11, the algorithm attempts to find paths of lower cost between intermediate vertices of the currently known path and the goal vertex. This step is necessary because the “patching” of subpaths in Line 17 can lead to suboptimal paths. However, in practical implementations, for the sake of speed Lines 9–11 need be executed only for high values of H , or they can be executed after completing the `while` loop of Line 4.

Incremental H -cost Motion-planning Algorithm

procedure MAIN

- 1: $H \leftarrow 0, n \leftarrow 0$
- 2: $\pi_0^0 \leftarrow \arg \min_{\pi \in \mathcal{L}_0(j_S, j_G)} \{\mathcal{J}_0(\pi)\}$
- 3: $H \leftarrow H + 1$
- 4: **while** $H \leq \bar{H}$ **do**
- 5: $P_n \leftarrow (\text{number of vertices of } \pi_n^0) - 1$
- 6: $I_k^H, k \in \{H, \dots, P_n\}$ defined as in Prop. 2
- 7: **if** $\mathcal{J}_H(\pi_n^0) < \chi$ **then**
- 8: **for** $k = P_n - 1, P_n - 2, \dots, 0$ **do**
- 9: **if** $\min_{\varsigma \in \mathcal{L}_H(I_k^H, I_{P_n}^H)} \{\mathcal{J}_H(\varsigma)\} < \sum_{\ell=k+1}^{P_n} \tilde{g}_H(I_{\ell-1}^H, I_{\ell}^H)$ **then**
- 10: $\pi_{n+1}^H \leftarrow$ path in \mathcal{G}_H obtained by replacing with $\arg \min_{\varsigma \in \mathcal{L}_H(I_k^H, I_{P_n}^H)} \{\mathcal{J}_H(\varsigma)\}$
 the edges between I_k^H and $I_{P_n}^H$ in path $b_0^H(\pi_n^0)$
- 11: $\pi_{n+1}^0 \leftarrow b_0^H(\pi_{n+1}^H)$
- 12: $n \leftarrow n + 1$
- 13: $H \leftarrow H + 1$
- 14: **else**
- 15: $k^* \leftarrow \arg \min \{k \in \{H + 1, \dots, P_n\} : \tilde{g}_H(I_{k-1}^H, I_k^H) = \chi\}$
- 16: $\{\zeta^*, \ell^*\} \leftarrow \text{PATH-REPAIR}(k^*)$
- 17: $\pi_{n+1}^H \leftarrow$ path in \mathcal{G}_H obtained by replacing with ζ^* the edges between $I_{k^*-1}^H$ and $I_{\ell^*}^H$
 in path $b_0^H(\pi_n^0)$
- 18: $\pi_{n+1}^0 \leftarrow b_0^H(\pi_{n+1}^H)$
- 19: $n \leftarrow n + 1$

procedure PATH-REPAIR(k^*)

- 1: **for** $\ell = k^*, \dots, P_n$ **do**
- 2: $\zeta^* \leftarrow \arg \min \{\varsigma \in \mathcal{L}_H(I_{k^*-1}^H, I_{\ell}^H)\}$
- 3: **if** $\mathcal{J}_H(\zeta^*) < \chi$ **then**
- 4: Return ($\zeta^*, \ell^* = \ell$)

Figure 4.2: Pseudocode description of the proposed incremental algorithm for solving the H -cost optimal path problem.

The optimization involved in Line 2 of the PATH-REPAIR procedure can be performed by an algorithm similar to the H -cost optimal path-planning algorithm described in [102].

The result of this algorithm is the path returned at either Line 11 or Line 18 of the final iteration of the algorithm. Whereas the algorithm terminates after a finite number of iterations (as we show next), its execution can be forcibly halted after Line 19 at any an intermediate iteration n before the algorithm's natural termination. Crucially, despite such a forcible termination, the algorithm returns a path π_n^0 in graph \mathcal{G} which is feasible for traversal (i.e. has H -cost less than χ), assuming such a path exists. This property of the algorithm of having ready a feasible path at the end of any iteration is highly desirable in real-time application.

The following result further highlights the merits of the proposed algorithm: not only does the algorithm eventually converge to an \bar{H} -cost optimal path, but the H -costs of paths found in intermediate iterations are almost always nonincreasing. Whenever H is incremented (Line 13), the transition costs of some edges in \mathcal{G}_{H+1} can possibly become higher than χ , and therefore the trend nonincreasing costs can be interrupted.

Proposition 3. *For a fixed value of the parameter H , whenever Lines 8–12 of Fig. 4.2 are executed, $\mathcal{J}_H(\pi_{n+1}^0) \leq \mathcal{J}_H(\pi_n^0)$, for each $n \in \mathbb{Z}_+$.*

Proof. Immediate by construction of π_{n+1}^0 in Fig. 4.2. □

Next, we state the main result of this chapter.

Proposition 4. *The proposed algorithm as described in Fig. 4.2 terminates after a finite number of iterations. Upon termination, the following statements hold true:*

- 1) *If, for some $\tilde{H} \leq \bar{H}$, there exists no path in graph \mathcal{G} with \tilde{H} -cost less than χ , then the algorithm reports failure.*
- 2) *If there exists at least one path in graph \mathcal{G} with \bar{H} -cost less than χ , then the algorithm returns a path with minimum \bar{H} -cost.*

Proof. Every iterative loop in the proposed algorithm in Fig. 4.2 has a finite termination condition. Therefore the overall algorithm must terminate in a finite number of iterations.

Case 1) Suppose for some $\tilde{H} \leq \bar{H}$, there exists no path in graph \mathcal{G} with H -cost less than χ and assume (without loss of generality) that \tilde{H} is the smallest such value of the parameter H . Suppose that the algorithm increments the value of the parameter H from $\tilde{H} - 1$ to \tilde{H} at the n^{th} iteration. Then $\mathcal{J}_{\tilde{H}}(\pi_n^0) \geq \chi$, and in the next iteration the algorithm executes Lines 7 and 15. At Line 15, the PATH-REPAIR must report failure because a subpath satisfying the condition in Line 3 of the PATH-REPAIR procedure does not exist. Then the overall algorithm also reports failure.

Case 2) Suppose there exists at least one path in graph \mathcal{G} with \bar{H} -cost less than χ . Then there exists a path in graph \mathcal{G} with minimum \bar{H} -cost, because the number of such paths is finite, in turn because the numbers of vertices and edges in \mathcal{G} are finite. According to the termination condition in Line 4, the overall algorithm terminates when $H = \bar{H}$. For any particular value of H , the final iteration of the `for` loop in Line 8 results in an H -cost optimal path. In particular, it results in the \bar{H} -cost optimal path. \square

Remark The proposed algorithm does not reduce the complexity of the H -cost optimal path problem. However, the proposed algorithm ensures that the search for an optimal path does not proceed in a “all or nothing” manner. Specifically, it makes available a feasible solution during intermediate iterations.

4.1 Illustrative Examples and Discussion

Figure 4.3 illustrates the execution of the proposed algorithm. Here, the prespecified initial and goal vertices are indicated, respectively, by the green- and red-colored vertices in Fig. 4.3(a). The path between the initial and final vertices highlighted in Fig. 4.3(a) is a 0-cost optimal path, i.e., it is found by solving the standard optimal path problem on \mathcal{G}_0 , where edge transition costs are defined as the Euclidean distances between cells. The algorithm

4.1. ILLUSTRATIVE EXAMPLES AND DISCUSSION

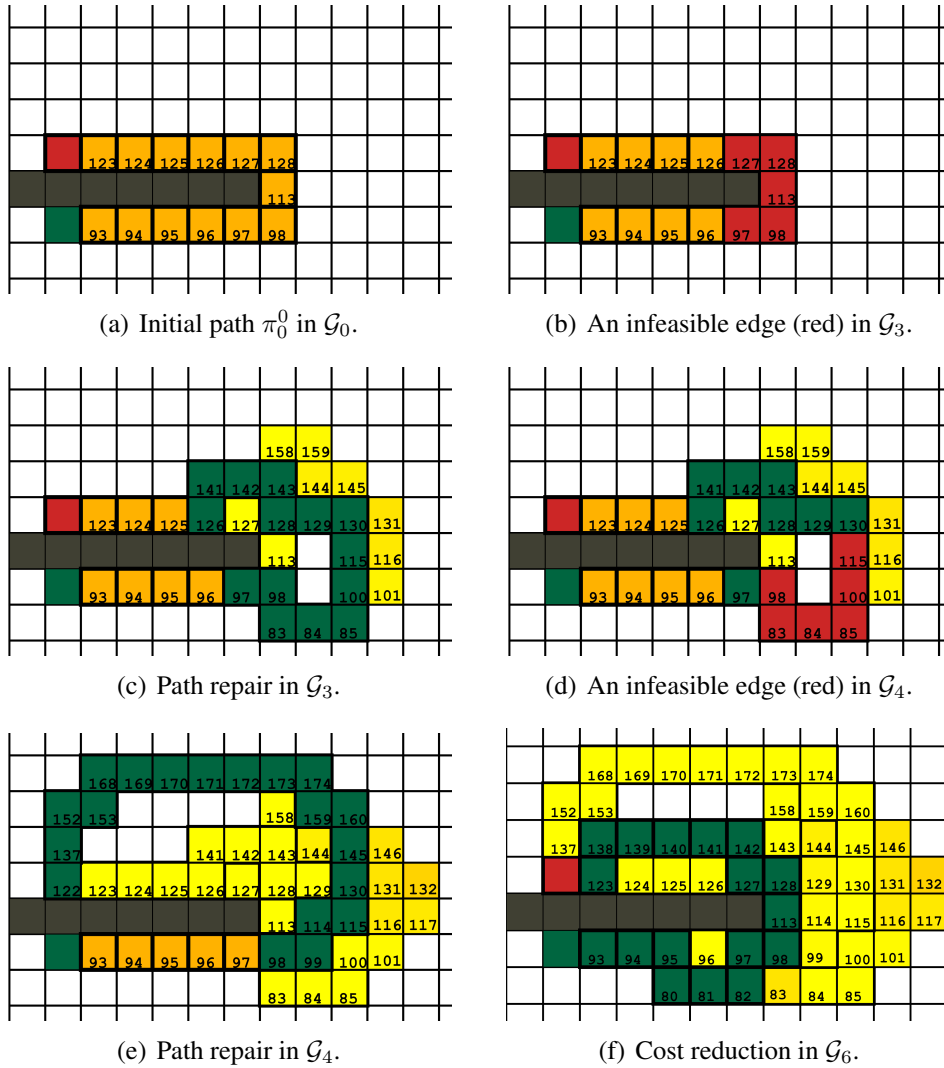


Figure 4.3: Illustrative example of solution of the H -cost optimal path problem using incremental path repair.

progresses to higher values of H , and when $H = 3$, the 3-cost of the path is found (Line 7 of the algorithm in Fig. 4.2) to be greater than or equal to χ . In particular, the transition cost of a single edge, illustrated in red in Fig. 4.3(b) is found to be equal to χ . In this example, H -costs are defined by Eqns. (2.7) and (2.8) for a Dubins car kinematic model with unit minimum turn radius. The edge with transition cost χ is replaced by the PATH-REPAIR procedure. The subpath ζ^* computed by the PATH-REPAIR procedure is indicated in green color in Fig. 4.3(c). Next, the 4-cost of the new path is found to be greater than or equal to χ (Fig. 4.3(d)) and it is again replaced by a subpath by the PATH-REPAIR procedure (Fig. 4.3(e)). The repaired path is found to have 6-cost less than χ , and at $H = 6$, the cost reduction steps in Lines 8-11 of the algorithm in Fig. 4.2 are executed to arrive at a 6-cost *optimal* path as shown in Fig. 4.3(f). The yellow-colored cells in Fig. 4.3 indicate the vertices explored by the PATH-REPAIR procedure during its computations to find a subpath (Line 16 in Fig. 4.2).

Figure 4.5 illustrates a more striking example of application the proposed algorithm. Here, a 4-cost feasible path is found quickly, and in further iterations, the algorithm finds paths of lower 4-cost. A feasible path is provided during intermediate iterations, and an optimal is eventually found.

4.1.1 o

In the geometric path-planning literature, anytime incremental algorithms such as LPA* [120] have been discussed for fast replanning when small changes in the environment are detected. The proposed incremental algorithm allows replanning H -cost optimal paths in response to such changes in the environment. Specifically, the current H -cost feasible or optimal path can become infeasible if changes in the environment cause a cell in the current path to be blocked by an obstacle. The proposed algorithm can detect such a change in Line 7 of Fig. 4.2.

Figure 4.4 illustrates an example of this replanning application of the proposed algo-

4.1. ILLUSTRATIVE EXAMPLES AND DISCUSSION

rithm. The initial and goal vertices are indicated in Fig. 4.4(a) by green- and red-colored cells. The black-colored cells indicate obstacles. As illustrated in Fig. 4.4(b), a change in the location of obstacles makes infeasible for traversal the initial path π_0^0 illustrated in Fig. 4.4(a). In Fig. 4.4(a), the algorithm has already progressed to $H = 1$, and due to the change in the environment, the 1-cost of the initial path is found to be higher than χ . The PATH-REPAIR procedure finds a subpath, indicated in green in Fig. 4.4(c). In a later iteration, the 3-cost of the new path is found higher than χ ; in particular one edge (indicated in red in Fig. 4.4(d)) is found to have cost χ . The PATH-REPAIR procedure finds a subpath in \mathcal{G}_3 to replace this edge, as indicated by the green-colored subpath in Fig. 4.4(e). The repaired path is found to have 6-cost less than χ and at $H = 6$, the cost reduction steps in Lines 8-11 of Fig. 4.2 are executed to arrive at a 6-cost optimal path as shown in Fig. 4.4(f).

Notice that in each of the two preceding examples, a 6-cost¹ *feasible* path is available in iterations *before* the optimal path is found. In real-time applications, this property can be used to enforce hard bounds on the computation time: the proposed algorithm will report a *feasible* path in the available computation time. By Prop. 4, if the algorithm is allowed sufficient computation time, it will converge to an optimal path, and by Prop. 3, the resultant H -costs are nonincreasing in all intermediate iterations except, possibly, in iterations where H is incremented.

¹In these particular examples, 6-cost feasible paths were found to be feasible for all higher values of H .

4.1. ILLUSTRATIVE EXAMPLES AND DISCUSSION

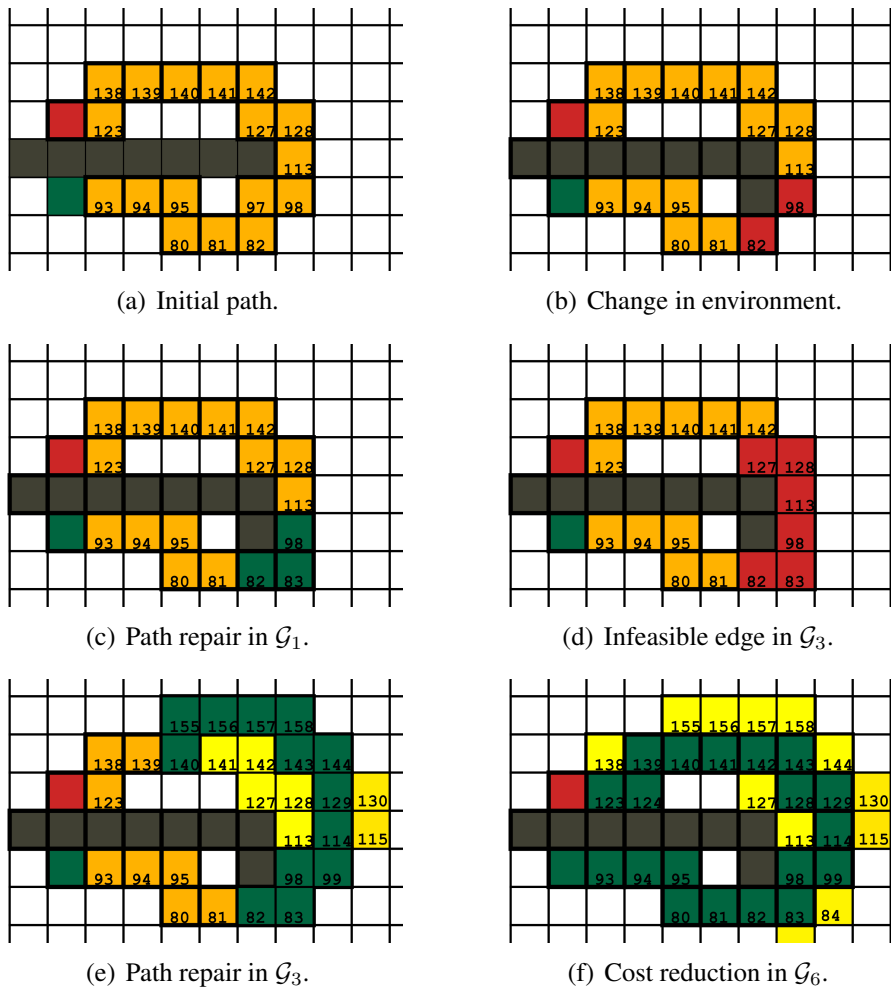
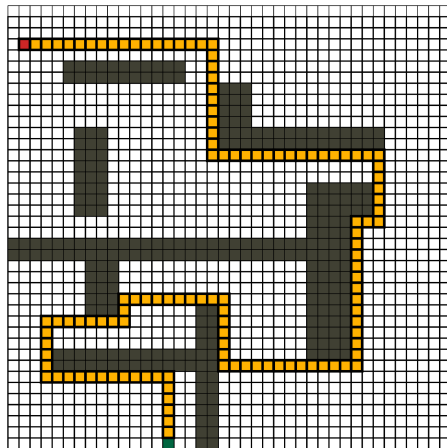
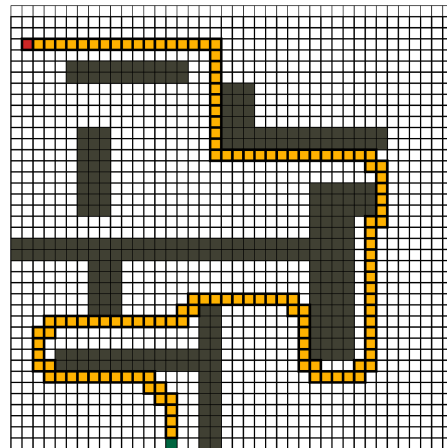


Figure 4.4: Illustration of replanning in response to changes in the environment.

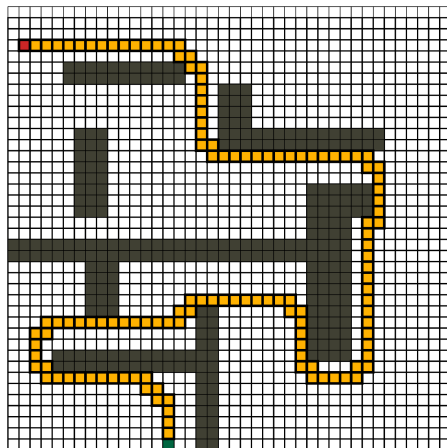
4.1. ILLUSTRATIVE EXAMPLES AND DISCUSSION



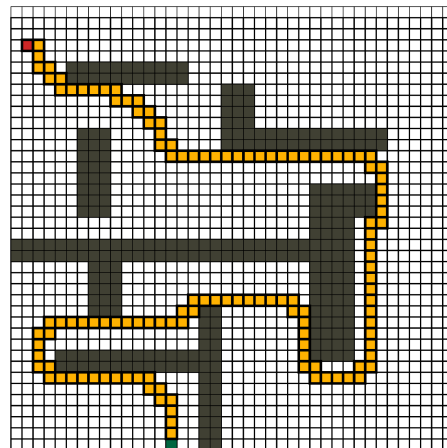
(a) Initial path.



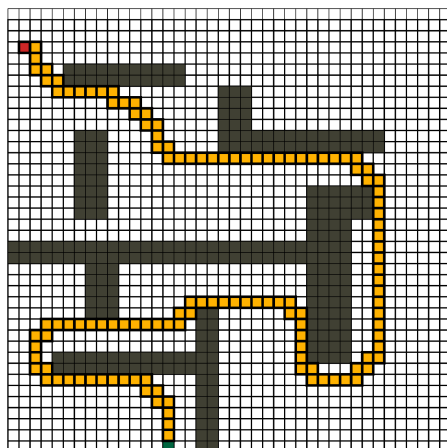
(b) $H = 1$ -cost feasible path.



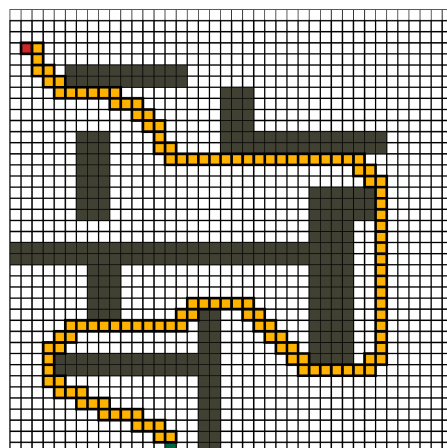
(c) $H = 4$ -cost feasible path.



(d) Reduced 4-cost feasible path.



(e) Reduced 4-cost feasible path.



(f) 4-cost optimal path.

Figure 4.5: Illustrative example: during intermediate iterations, feasible solutions are available, whereas the solution cost is reduced.

Chapter 5

Sampling Based Motion-Planning

In this chapter, we propose a randomized sampling-based motion planning algorithm to satisfy LTL specifications. Similar to RRT*, the proposed algorithm incrementally generates a tree structure representative of the vehicle dynamical system model. Random samples are taken from multiple copies of the state space, with each copy uniquely associated with a state of the Büchi automaton that accepts the given LTL specification. The proposed algorithm retains the properties of RRT*: namely, the proposed algorithm is probabilistically complete (i.e. guaranteed to find a trajectory satisfying the specifications if it exists) and asymptotically optimal. To achieve significant reductions in computation time, we propose a sampling heuristic that provides a bias for growing the tree structure. We provide numerical simulation results of the application of the proposed algorithm to a vehicle kinematic model same as in chapter 2 and to a quadrotor aircraft dynamical model.

5.1 Problem Formulation

The workspace is defined the same as in chapter 2, and it is assumed to have *obstacles*, i.e. regions that are forbidden for the vehicle to enter. We denote by \mathcal{W}_{obs} the obstacle-occupied region of the workspace; the obstacle-free workspace is then $\mathcal{W}_{\text{free}} = \mathcal{W} \setminus \mathcal{W}_{\text{obs}}$.

We consider a vehicle kinematic model described by the differential equations as in chapter 2

$$\dot{p}_x(t) = \cos \psi(t), \quad \dot{p}_y(t) = \sin \psi(t), \quad \dot{\psi}(t) = u(t), \quad (5.1)$$

5.2. LTL SATISFACTION BY STATE TRAJECTORIES

where the steering rate u is the control input and ψ is the heading angle. We assume $u(t) \in U := \left[\frac{1}{\rho}, \frac{1}{\rho}\right]$ for all $t \in \mathbb{R}$, where $\rho > 0$ is a prespecified constant. Note that ρ is the normalized minimum radius of turn. The state variable is $\mathbf{x} = (\mathbf{p}, \psi) \in \mathcal{D} = \mathcal{W} \times [0, 2\pi]$. The obstacle-free state space is defined as $\mathcal{D}_{\text{free}} := \mathcal{W}_{\text{free}} \times [0, 2\pi]$.

A rigid-body dynamical model of quadrotor aircraft motion is described by

$$\begin{bmatrix} \dot{\psi} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix} = \frac{1}{\cos \theta} \begin{bmatrix} 0 & \sin \phi & \cos \phi \\ 0 & \cos \phi \cos \theta & -\sin \phi \cos \theta \\ \cos \theta & \sin \phi \sin \theta & \cos \phi \sin \theta \end{bmatrix} \boldsymbol{\Omega}, \quad (5.2)$$

$$m\ddot{\mathbf{p}} = -g\hat{\mathbf{k}} + \mathbf{u}_{[1]}R_b^i\hat{\mathbf{k}}, \quad I\dot{\boldsymbol{\Omega}} = \mathbf{u}_{[2:4]} + \boldsymbol{\Omega} \times J\boldsymbol{\Omega}. \quad (5.3)$$

Here m is the vehicle mass; $\hat{\mathbf{k}} = (0, 0, 1)$ is a unit basis vector; ψ, θ, ϕ are yaw, pitch, and roll angles, respectively in the standard 3-2-1 Euler angle system; R_b^i is the corresponding rotation matrix to transform coordinates from body-fixed axes to inertial axes; I is the vehicle's mass moment of inertia matrix; and $\boldsymbol{\Omega}$ is the inertial angular velocity vector with coordinates in a body-fixed axes system. The convention (cf., [121, 122]) is to consider control inputs $\mathbf{u}(t) \in \mathbb{R}^4$, where $\mathbf{u}_{[1]}$ is the magnitude of the total thrust generated by the rotors, and $\mathbf{u}_{[2:4]}$ are body-fixed axes coordinates of the total moment at the vehicle c.m. due to the rotors. The state variable is $\mathbf{x} = (\mathbf{p}, \dot{\mathbf{p}}, \psi, \theta, \phi, \boldsymbol{\Omega}) \in \mathcal{D} = \mathcal{W} \times \mathbb{R}^3 \times [0, 2\pi]^3 \times \mathbb{R}^3$. The obstacle-free state space is defined as $\mathcal{D}_{\text{free}} := \mathcal{W}_{\text{free}} \times \mathbb{R}^3 \times [0, 2\pi]^3 \times \mathbb{R}^3$. The attitude dynamical equations (5.2) can be alternatively written using quaternions [122].

For these models, we denote by $\mathbf{x}(\cdot; \mathbf{x}_0, \mathbf{u})$ the state trajectory $t \mapsto \mathbf{x}(t)$ with initial condition \mathbf{x}_0 and control input $t \mapsto \mathbf{u}(t)$ for $t > 0$.

5.2 LTL Satisfaction by State Trajectories

To formulate LTL specifications of relevance to mobile vehicles, it is convenient to define a finite number of regions of interest in the workspace $\lambda_1, \dots, \lambda_N \in \mathcal{W}_{\text{free}}$. We can

associate with each of these regions a unique atomic proposition defined to be true whenever the location of the vehicle's c.m. \mathbf{p} is within the region, i.e. the k^{th} proposition is true whenever $\mathbf{p} \in \lambda_k$. We label these atomic propositions also by $\lambda_1, \dots, \lambda_N$, and define $\Lambda := \{\lambda_1, \dots, \lambda_N\}$. Informally, each LTL formula is associated with a *sequence* of regions to be visited in the workspace. A vehicle state trajectory *satisfies* the LTL formula if it sequentially passes through these regions.

Next, let $\Gamma : \mathcal{D}_{\text{free}} \rightarrow 2^\Lambda$ be a map from each state in $\mathcal{D}_{\text{free}}$ to a set of atomic propositions that are true at that state. To be precise, for the state $\mathbf{x} = (\mathbf{p}, \dots) \in \mathcal{D}_{\text{free}}$, $\Gamma(\mathbf{x}) := \cup_\ell \lambda_\ell$ for each $\ell \in \{1, \dots, N\}$ satisfying $\mathbf{p} \in \lambda_\ell$. Next, for a trajectory $\mathbf{x}(t; \mathbf{x}_0, \mathbf{u})$ of the vehicle model, let $0 < t_1 < t_2 < \dots$ be a sequence of time instants such that $\Gamma(\mathbf{x}(t_m; \mathbf{x}_0, \mathbf{u})) \neq \Gamma(\mathbf{x}(t_{m+1}; \mathbf{x}_0, \mathbf{u}))$ for each integer $m \geq 0$. The trajectory $\mathbf{x}(t; \mathbf{x}_0, \mathbf{u})$ is said to define the word $\bar{\omega}(\mathbf{x}(t; \mathbf{x}_0, \mathbf{u})) := (\omega_0, \omega_1, \dots)$ where $\omega_m := \Gamma(\mathbf{x}(t_m; \mathbf{x}_0, \mathbf{u}))$ for each integer $m \geq 0$. Finally, the trajectory $\mathbf{x}(t; \mathbf{x}_0, \mathbf{u})$ *satisfies* the LTL formula ϕ if $\bar{\omega}(\mathbf{x}(t; \mathbf{x}_0, \mathbf{u})) \models \phi$.

An accepting run of \mathcal{B}_ϕ can be infinitely long. This situation occurs whenever the LTL formula includes the compound operator $\square\lozenge$ (*always eventually*), which indicates a requirement of infinitely persistent visits to one or more region in the workspace. This situation can be problematic for defining appropriate cost functionals for the vehicle state trajectory. Several works in the literature (cf. [123]) avoid this issue by addressing a subset of LTL specifications called *co-safe* specifications [2]. Informally, co-safe LTL formulae are defined by excluding the *always* \square operator. Infinitely long runs can be found in a *prefix-suffix* form, where the prefix and suffix are each finite paths in the product system. The suffix begins and ends at an accepting state of \mathcal{B}_ϕ and loops infinitely often.

The proposed algorithm does not restrict LTL specifications to the co-safe class. To define a finite state trajectory cost (without artificially introducing a discount factor), we consider the cost of the state trajectory only up to the first suffix. To be precise, let $\mathbf{x}(t; \mathbf{x}_0, \mathbf{u})$ be a state trajectory satisfying the LTL specification ϕ . If ϕ is co-safe, then there exists a finite t_f for which the restricted state trajectory $\mathbf{x}(t; \mathbf{x}_0, \mathbf{u})|_{[0, t_f]}$ also satisfies ϕ . If ϕ is not co-safe, then there exist $t_{f,p}$ and $t_{f,s}$ such that the restricted trajectories $\mathbf{x}(t; \mathbf{x}_0, \mathbf{u})|_{[0, t_{f,p}]}$ and

$\mathbf{x}(t; \mathbf{x}_0, \mathbf{u})|_{[t_{f,p}, t_{f,s}]}$ are associated with the prefix and the first suffix subpaths of the (infinitely long) path of the product transition system that satisfies ϕ . In this case, we define $t_f := t_{f,p} + t_{f,s}$. In either case, the cost of the state trajectory is $J(\mathbf{u}) := \int_0^{t_f} g(\mathbf{x}(t; \mathbf{x}_0, \mathbf{u})) dt$, where $g : \mathcal{D} \rightarrow \mathbb{R}^{\geq 0}$ is a nonnegative cost function.

Problem 3. *Given an initial state $\mathbf{x}_0 \in \mathcal{D}$ and a LTL formula ϕ over Λ , find a control input $\mathbf{u}^* : [0, \infty) \mapsto U$ such that the trajectory $\mathbf{x}(t; \mathbf{x}_0, \mathbf{u}^*)$ (1) avoids obstacles, i.e. $\mathbf{x}(t; \mathbf{x}_0, \mathbf{u}^*) \in \mathcal{D}_{\text{free}}$ for all $t \in [0, \infty)$, (2) satisfies the specification ϕ , i.e. $\bar{\omega}(\mathbf{x}(t; \mathbf{x}_0, \mathbf{u}^*)) \models \phi$, and (3) minimizes the trajectory cost, i.e., $J(\mathbf{u}^*) \leq J(\mathbf{u})$ for all piecewise continuous functions $\mathbf{u} : [0, \infty) \rightarrow U$.*

5.3 Proposed Solution to Problem 3

In this section, we discuss the proposed incremental sampling-based algorithm to solve Problem 3 with a probabilistic guarantee of completeness and with asymptotic optimality. The given LTL formula is first translated to a Büchi automaton $\mathcal{B}_\phi = (S, s_0, \Sigma, \delta_{\mathcal{B}}, F)$. We can then define the *product space* $T := \mathcal{D} \times S$. The main algorithm is similar to RRT* [76]. The primary difference is that in the proposed algorithm random samples are taken from the product space T and sampling is biased for computational efficiency. The tree constructed from these samples then serves as the product transition system. Algorithm 5.1 provides a precise description of the proposed approach. The descriptions of subroutines used by the algorithm are provided in Section 5.3.1.

During its execution, Algorithm 5.1 maintains a tree¹ $\mathcal{G} = (V, E)$. Initially, this tree includes the initial product state $\theta_0 = (\mathbf{x}_0, s_0)$ as its root vertex in V and no edges. There are two vertex sets V_{prefix} and V_{suffix} associated with states for searching prefix and suffix paths, respectively. For each vertex $\theta \in V = V_{\text{prefix}} \cup V_{\text{suffix}}$, the algorithm associates a scalar cost of reaching this vertex from the root θ_0 ; this cost is denoted by $Cost(\theta)$. We denote states in the product space T and also the associated vertex in V by the same symbol θ .

¹A tree is a graph with no cycles.

Sampling based motion planning algorithm to satisfy LTL specifications

```

1:  $V_{\text{prefix}} \leftarrow \{\theta_0 = (\mathbf{x}_0, s_0)\}$ ,  $V_{\text{suffix}} \leftarrow \emptyset$ ,  $E \leftarrow \emptyset$ ,  $i \leftarrow 0$ ,  $Cost(\theta_0) \leftarrow 0$ 
2: while  $i < N$  do
3:    $i = i + 1$ 
4:    $\theta_{\text{rnd}} \leftarrow \text{Sample}(V_{\text{prefix}})$ ,  $\theta_{\text{nr}} \leftarrow \text{Nearest}(V_{\text{prefix}}, \theta_{\text{rnd}})$ ,  $\theta_{\text{new}} \leftarrow \text{Steer}(\theta_{\text{nr}}, \theta_{\text{rnd}})$ 
5:   if  $\text{FeasCheck}(\theta_{\text{nr}}, \theta_{\text{new}})$  then
6:      $V_{\text{prefix}} \leftarrow V_{\text{prefix}} \cup \{\theta_{\text{new}}\}$ 
7:      $(V_{\text{prefix}}, E) \leftarrow \text{UpdateTree}(V_{\text{prefix}}, E, \theta_{\text{new}})$ 
8:     if  $\theta_{\text{new}.s \in F$  then
9:        $V_{\text{suffix}} = V_{\text{suffix}} \cup \{\theta_{\text{new}}\}$ 
10:       $(V_{\text{suffix}}, E) \leftarrow \text{UpdateTree}(V_{\text{suffix}}, E, \theta_{\text{new}})$ 
11:       $V \leftarrow V_{\text{prefix}} \cup V_{\text{suffix}}$ 
12:       $\theta_{\text{rnd}} \leftarrow \text{Sample}(V_{\text{suffix}})$ ,  $\theta_{\text{nr}} \leftarrow \text{Nearest}(V_{\text{suffix}}, \theta_{\text{rnd}})$ ,  $\theta_{\text{new}} \leftarrow \text{Steer}(\theta_{\text{nr}}, \theta_{\text{rnd}})$ 
13:      if  $\text{FeasCheck}(\theta_{\text{nr}}, \theta_{\text{new}})$  then
14:         $V_{\text{suffix}} \leftarrow V_{\text{suffix}} \cup \{\theta_{\text{new}}\}$ 
15:         $(V_{\text{suffix}}, E) \leftarrow \text{UpdateTree}(V_{\text{suffix}}, E, \theta_{\text{new}})$ 
16:         $V \leftarrow V_{\text{prefix}} \cup V_{\text{suffix}}$ 
17:         $V_{\text{nearby}} \leftarrow \text{Near}(V_{\text{prefix}} \cup V_{\text{suffix}}, \theta_{\text{new}})$ 
18:        for all  $\theta \in V_{\text{nearby}}$  do
19:           $\text{SuffixCheck}(\theta_{\text{new}}, \theta)$ 

```

Figure 5.1: Pseudocode description of the proposed sampling based algorithm for solving the task/path planning problem.

At each iteration, the algorithm expands the tree to search for a prefix path (Line 4-12 in Alg. 5.1), and separately expands the tree to search for a suffix path (Line 13-19 in Alg. 5.1). The procedure for expanding the tree for the prefix path is as follows. A new sample θ_{rnd} is generated (Line 4 in Alg. 5.1), and a vertex in V_{prefix} nearest to θ_{rnd} is found. The subroutine *Steer* is used to extend the tree to a new state θ_{new} towards θ_{nr} . After checking the feasibility (Line 5 in Alg. 5.1, discussed in Section 5.3.1) of the trajectory from θ_{nr} to θ_{new} returned by the *Steer* subroutine, the state θ_{new} is inserted as a new vertex into V_{prefix} . The algorithm then calls the subroutine *UpdateTree* to connect θ_{new} by a minimum-cost trajectory and rewires the edges in the tree (see Alg. 5.2).

If $\theta_{\text{new}.s}$ is an accepting state, i.e., $\theta_{\text{new}.s} \in F$, then θ_{new} is inserted as a new vertex into V_{suffix} and the tree is updated (Lines 9 and 10 in Alg. 5.1). The procedure for expanding the tree for the suffix path is similar (Lines 13-19 in Alg. 5.1). The suffix path is a cycle that begins and ends at an accepting state of \mathcal{B}_ϕ . Since a tree does not have cycles by definition, *SuffixCheck* checks if a cycle can be constructed by adding an edge between two existing vertices. Subroutines *Steer* and *FeasCheck* ensure that all potential edges of the tree are feasible transitions of \mathcal{B}_ϕ . These subroutines also ensure that the trajectory between any two states is obstacle-free and is generated by an admissible control input.

5.3.1 Description of Subroutines

Subroutine *Sample*(V) returns $\theta = (\theta.\mathbf{x}, \theta.s) \in V$, where $\theta.\mathbf{x}$ is sampled from a uniform distribution over the state space \mathcal{D} , and $\theta.s$ is sampled from a uniform distribution over the subset $S_{\text{rch}} := \cup_{\theta \in V} \{\theta.s\} \subseteq S$.

Subroutine *Dist*($\mathbf{x}_1, \mathbf{x}_2$) returns the cost of an optimal trajectory between \mathbf{x}_1 and \mathbf{x}_2 , i.e., $\text{Dist}(\mathbf{x}_1, \mathbf{x}_2) := \min_{\mathbf{u}: [0, t_2] \rightarrow U} J(\mathbf{u})$, such that $\mathbf{x}(t_2; \mathbf{x}_1, \mathbf{u}) = \mathbf{x}_2$.

Subroutine *Steer*(θ_1, θ_2) first finds $\mathbf{x}_{\text{new}} \in \mathcal{D}$ such that $\text{Dist}(\theta_1.\mathbf{x}, \mathbf{x}_{\text{new}}) < \eta$, for a prespecified constant $\eta > 0$ and such that $\text{Dist}(\mathbf{x}_{\text{new}}, \theta_2.\mathbf{x})$ is minimum. Second, *Steer* finds $s_{\text{new}} \in S$ in the Büchi automaton such that $s_{\text{new}} = \delta_{\mathcal{B}}(\theta_1.s, \Gamma(\mathbf{x}_{\text{new}}))$. Recall that $\Gamma(\mathbf{x})$ is the

5.3. PROPOSED SOLUTION TO PROBLEM 3

set of atomic propositions associated with a state transition of the Büchi automaton. *Steer* returns $\theta_{\text{new}} = (\mathbf{x}_{\text{new}}, s_{\text{new}})$.

Subroutine *FeasCheck*(θ_1, θ_2) checks if there exists a control input $\mathbf{u} : [0, t_2] \mapsto U$ such that $\mathbf{x}(t; \theta_1.\mathbf{x}, \mathbf{u}) \in \mathcal{D}_{\text{free}}$ for all $t \in [0, t_2]$, and $\mathbf{x}(t_2; \theta_1.\mathbf{x}, \mathbf{u}) = \theta_2.\mathbf{x}$, where $t_2 > 0$. The subroutine also *model checks*, i.e., determines whether $(\theta_1.s, \theta_2.s) \in \delta_{\mathcal{B}}$. If both these two conditions are true, the function returns `true`, otherwise it returns `false`.

The subroutine *Dist*, *Steer*, and *FeasCheck* are all involved in forming edges in the tree \mathcal{G} , associating these edges with admissible control inputs for the vehicle kinematical or dynamical model, and ensuring that the resulting vehicle state trajectory is obstacle-free. Therefore, implementations of Alg. 5.1 can take advantage of the functional overlaps among these subroutines and avoid repeated trajectory optimization computations. These subroutines involve the solution of the familiar trajectory optimization problem of finding an admissible control input from \mathbf{x}_1 to \mathbf{x}_2 in \mathcal{D} . Because these subroutines are repeatedly invoked, the solution to this problem must be computationally efficient. Fortunately, the literature provides efficient trajectory optimization solutions for the two aircraft models discussed in Section 2.1.

For the aircraft kinematical model, minimum-time trajectories are exactly found by the well-known Dubins paths constructions [40], which we implement. For the quadrotor dynamical model, we implement the minimum-snap trajectory optimization method discussed by Mellinger & Kumar [121]. The details of these methods are found in the stated references, and are omitted here. If vehicle kinematical/dynamical constraints are ignored, then $\mathcal{D} = \mathcal{W}$, and the optimal trajectory between any two states $\mathbf{x}_1, \mathbf{x}_2$ is simply a straight line.

Subroutine *Nearest*(V, θ) returns $\theta_{\text{nr}} \in V$ such that $\theta_{\text{nr}}.s = \theta.s$ where $\text{Dist}(\theta_{\text{nr}}.\mathbf{x}, \theta.\mathbf{x})$ is minimum. Subroutine *Near*(V, θ) returns the set $V \cap \{v \in T \mid \text{Dist}(v.\mathbf{x}, \theta.\mathbf{x}) < \eta \text{ and } \text{Dist}(\theta.\mathbf{x}, v.\mathbf{x}) < \eta\}$ for a prespecified $\eta > 0$.

Subroutine *SuffixCheck*(θ_1, θ_2) first executes *FeasCheck*(θ_1, θ_2). If *FeasCheck*(θ_1, θ_2) is true and if an edge between θ_1 and θ_2 results in a cycle in G , then *SuffixCheck* returns

5.3. PROPOSED SOLUTION TO PROBLEM 3

Subroutine $UpdateTree(V, E, \theta_{new})$

```

1:  $\theta_{newpr} \leftarrow Nearest(V, \theta_{new})$ 
2:  $c_{new} \leftarrow Cost(\theta_{newpr}) + Dist(\theta_{newpr} \cdot \mathbf{x}, \theta_{new} \cdot \mathbf{x})$ 
3:  $V_{nearby} \leftarrow Near(V, \theta)$ 
4: for all  $\theta_{near} \in V_{nearby}$  do
5:   if  $FeasCheck(\theta_{near}, \theta_{new})$  then
6:      $c' \leftarrow Cost(\theta_{near}) + Dist(\theta_{near} \cdot \mathbf{x}, \theta_{new} \cdot \mathbf{x})$ 
7:     if  $c' < c_{new}$  then
8:        $\theta_{newpr} \leftarrow \theta_{near}; c_{new} \leftarrow c'$ 
9:    $E \leftarrow E \cup \{(\theta_{newpr}, \theta_{new})\}$ 
10: for all  $\theta_{near} \in V_{nearby} \setminus \{\theta_{newpr}\}$  do
11:   if  $FeasCheck(\theta_{new}, \theta_{near})$  and  $Cost(\theta_{near}) > Cost(\theta_{new}) + Dist(\theta_{new} \cdot \mathbf{x}, \theta_{near} \cdot \mathbf{x})$ 
   then
12:      $\theta_{parent} \leftarrow Parent(\theta_{near})$ 
13:      $E \leftarrow (E \setminus \{(\theta_{parent}, \theta_{near})\}) \cup \{(\theta_{new}, \theta_{near})\}$ 
14:      $Cost(\theta_{near}) \leftarrow Cost(\theta_{new}) + Dist(\theta_{new} \cdot \mathbf{x}, \theta_{near} \cdot \mathbf{x})$ 
15: return  $(V, E)$ 

```

Figure 5.2: Pseudocode description of the Update random tree subroutine.

true. If there is a vertex θ in this cycle such that $\theta.s \in F$, then this cycle is a suffix.

Subroutine $UpdateTree(V, E, \theta_{new})$ reassigns some of the edges in this tree \mathcal{G} . In Alg. 5.2, the state $\theta_{newpr} \in Near(V, \theta_{new})$ that can be steered to θ_{new} such that $Cost(\theta_{new})$ becomes minimum, is chosen as the parent to θ_{new} (Lines 4-9 in Alg. 5.2). This means that the pair $(\theta_{newpr}, \theta_{new})$ is added to the edge set E (Line 9 in Alg. 5.2). The next loop (Line 10-13 in Alg. 5.2) attempts to connect θ_{new} to states that are already in V . For any state $\theta_{near} \in Near(V, \theta_{new})$, if a trajectory from $\theta_{new} \cdot \mathbf{x}$ to $\theta_{near} \cdot \mathbf{x}$ exists such that $Cost(\theta_{near})$ is lowered (Line 11 in Alg. 5.2), then θ_{new} is assigned the new parent of θ_{near} (Lines 12 and 13 in Alg. 5.2). Note that $Cost(\theta)$ is recursively defined in Alg. 5.2 (Lines 2, 6, and 14 in Alg. 5.2) for each vertex $\theta \in V$.

Heuristic Sampling Subroutine

```

1: Initialize  $S_{\text{rch}}, S_{\text{ext}}$ 
2:  $s_{\text{rnd}} \leftarrow \text{Rand}(S)$ 
3: if  $s_{\text{rnd}} \in S_{\text{rch}}$  then
4:    $\mathbf{x}_{\text{rnd}} \leftarrow \text{Rand}(\mathcal{D})$ 
5: else
6:    $s_{\text{ext}} \leftarrow \text{Rand}(S_{\text{ext}})$ , and let  $\lambda \in \Sigma$  be such that  $(s_{\text{rnd}}, \lambda, s_{\text{ext}}) \in \delta_{\mathcal{B}}$ 
7:    $\mathcal{D}_{\text{sample}} = \{\mathbf{x} \mid \Gamma(\mathbf{x}) = \lambda\}$ , and  $\mathbf{x}_{\text{rnd}} \leftarrow \text{Rand}(\mathcal{D}_{\text{sample}})$ 
8: return  $\theta_{\text{rnd}} = (s_{\text{rnd}}, \mathbf{x}_{\text{rnd}})$ 

```

Figure 5.3: Pseudocode description of sampling heuristic subroutine.

5.3.2 Sampling Heuristic

To speed up the execution of Alg. 5.1, we propose the following sampling heuristic (Alg. 5.3). Recall that the subroutine $\text{Sample}(V)$ samples a state from the set $S_{\text{rch}} = \cup_{\theta \in V} \{\theta.s\}$. We define the set

$$S \supseteq S_{\text{ext}} := \{s \mid (s', s) \in \delta_{\mathcal{B}} \text{ and } s' \in S_{\text{rch}}\} \setminus S_{\text{rch}}.$$

Informally, S_{ext} is the set of states in the Büchi automaton that can be reached by a single transition from any state of S_{rch} , but are not already in S_{rch} .

In Line 2 of Alg.5.3, the subroutine Rand returns a sample s_{rnd} from a uniform distribution over S (the set of all states of the Büchi automaton). If $s_{\text{rnd}} \in S_{\text{rch}}$, then the sampling strategy is the same as in the aforementioned Sample subroutine. Otherwise, the set S_{ext} is sampled (Line 6).

Intuitively, this sampling subroutine sets a sampling bias whenever $S_{\text{rch}} \neq S$. This sampling bias preserves the properties of probabilistic completeness and asymptotic optimality of Alg. 5.1, which are inherited from the RRT* algorithm. With increasing iterations of Alg. 5.1, the set S_{rch} becomes larger, until $S_{\text{rch}} = S$. Once this condition occurs (recall that S is a finite set), the sampling heuristic in Alg. 5.3 becomes identical to the Sample subrou-

tine, for which the aforesaid properties of completeness and optimality are proven [76]. The significant speed-up due to this sampling heuristic is discussed in Section 5.4.

The probabilistic completeness and exponential convergence of RRT* have been proven in [124]. The proposed algorithm inherits these properties. The proposed algorithm also inherits the asymptotic optimality property of RRT*. This is because it rejects a new edge only if the result of the *Steer* subroutine fails the model check, in which case the edge is not associated with a transition of the Büchi automaton.

5.4 Numerical Simulation Examples

We implemented the algorithms presented in this paper with C++. All simulations discussed in this section were performed on a desktop computer with a 3.4 GHz Intel Core i7 processor, 8GB of memory, and the Ubuntu 14.04 operating system. The open-source library Spot 2.0 was used to generate Büchi automata from LTL specifications [125]. Open-source code for the proposed work is available at the following repository: https://github.com/zetian/ltl_sampling.

5.4.1 Example Neglecting Kinematical Constraints

Consider a 100×100 unit square size environment as shown in Fig. 5.4(a). The initial position is at $(30, 30)$, and the task assigned to the vehicle is to visit regions $\lambda_1, \lambda_2, \lambda_3$ and λ_4 (orange regions in Fig. 5.4(a)) infinitely many times while avoiding obstacles (gray regions). The LTL specification associated with this task is $\phi = \square(\diamond\lambda_1 \wedge \diamond\lambda_2 \wedge \diamond\lambda_3 \wedge \diamond\lambda_4)$. A vehicle kinematical or dynamical model is not considered.

A solution to this problem is shown in Fig. 5.4(a) marked as a black path. Figure 5.4(b) shows the tree maintained by the algorithm. The proposed algorithm was repeatedly executed for 20 times on this example. The average execution time to find the first feasible solution was less than 0.1 s. By contrast, the average execution time to find the first feasible solution without the proposed sampling heuristic was 0.87 sec. In other words, proposed sampling

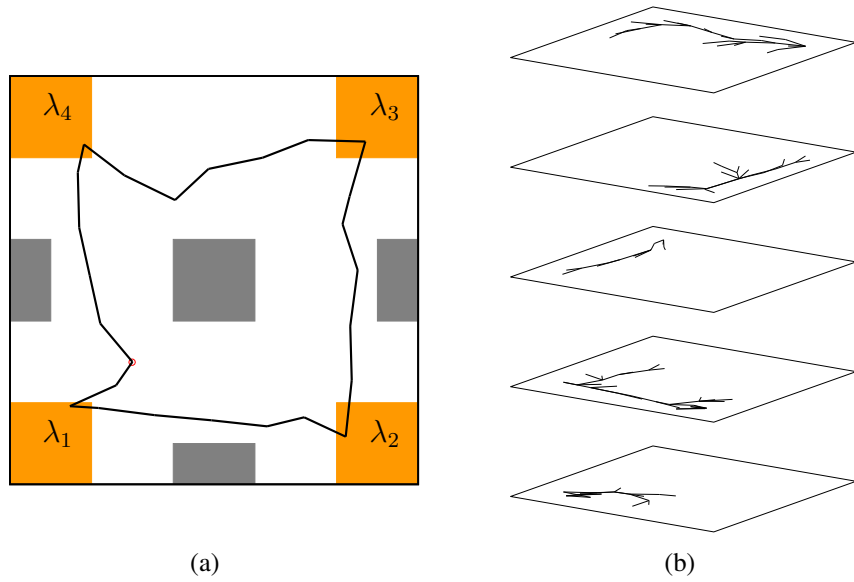


Figure 5.4: Workspace (a) and product space (b) for the example in Section 5.4.1.

heuristic resulted in over 75% reduction in the average execution time to find the first feasible solution.

Next, we evaluated the performance advantage due to the proposed sampling heuristic via a series of simulations with three different LTL specifications and with different obstacle locations in each simulation. The results of this study are shown in Fig. 5.5. The horizontal axis of the plot in Fig. 5.5 is the ratio of the area occupied by obstacles to the area of the entire workspace, i.e., $\text{Area}(\mathcal{W}_{obs})/\text{Area}(\mathcal{W})$. The vertical axis is the ratio of the execution times of the proposed algorithm to find a first feasible solution with and without the proposed sampling heuristic. Note that when $\text{Area}(\mathcal{W}_{obs})/\text{Area}(\mathcal{W})$ is low, the sampling heuristic demonstrates a significant reduction in execution time: approximately an 80% reduction depending on the number of states in the Büchi automaton associated with the LTL specification. Even as $\text{Area}(\mathcal{W}_{obs})/\text{Area}(\mathcal{W})$ increases, a performance advantage due to the proposed sampling heuristic is obtained. For higher values of $\text{Area}(\mathcal{W}_{obs})/\text{Area}(\mathcal{W})$, i.e., when the workspace is too densely cluttered with obstacles, the well-known “narrow channel” problem severely impedes randomized sampling-based algorithms. Each data point in Fig. 5.5 is an average value over 20 simulation instances.

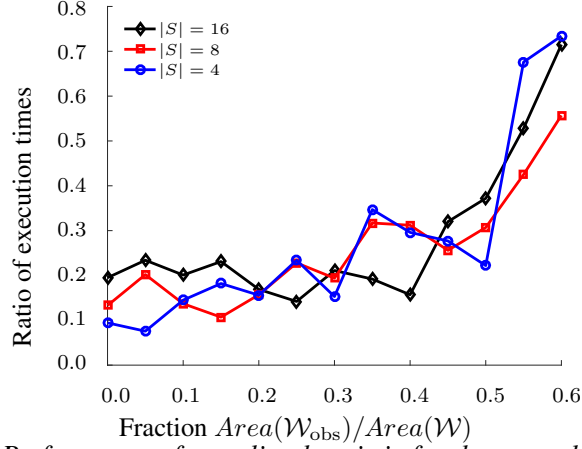


Figure 5.5: Performance of sampling heuristic for the example in Section 5.4.1.

5.4.2 Examples with Fixed-Wing Aircraft Model

Example 1. Here we consider an example with the fixed wing aircraft kinematical model (5.1). The task assigned to the vehicle is to visit regions $\lambda_1, \lambda_2, \lambda_3$ (indicated by shaded regions in 5.7(a)) with no imposition on the order of visits, i.e., the LTL specification is $\phi_1 = \diamond \lambda_1 \wedge \diamond \lambda_2 \wedge \diamond \lambda_3$. The size of the workspace is 100×100 unit square, and the minimum turning radius of the vehicle model $\rho = 15$ units. The initial state is $(50, 10, \pi/2)$, indicated by a small circle. The trajectories resulting from the application of the proposed algorithm after 600, 1200, and 3000 iterations with execution times 1.43 s, 3.29 s, and 14.72 s are shown in Fig. 5.7(a), Fig. 5.7(c), and Fig. 5.7(e), respectively. Figures 5.7(b), 5.7(d), and 5.7(f) illustrate the tree \mathcal{G} at these iterations.

As the number of iterations increase, the costs of trajectories returned by the proposed algorithm decrease. For this example, decreasing trajectory costs with increasing iterations of the algorithm are shown in Fig. 5.6(a).

Similar to Fig. 5.5, Fig. 5.6 shows the computational advantage provided by the proposed sampling heuristic for Example 1. The horizontal and vertical axes in Fig. 5.6 are the same as in Fig. 5.5. the sampling heuristic provides reductions in execution time by as much as 80% depending on the number of states in the Büchi automaton.

5.4. NUMERICAL SIMULATION EXAMPLES

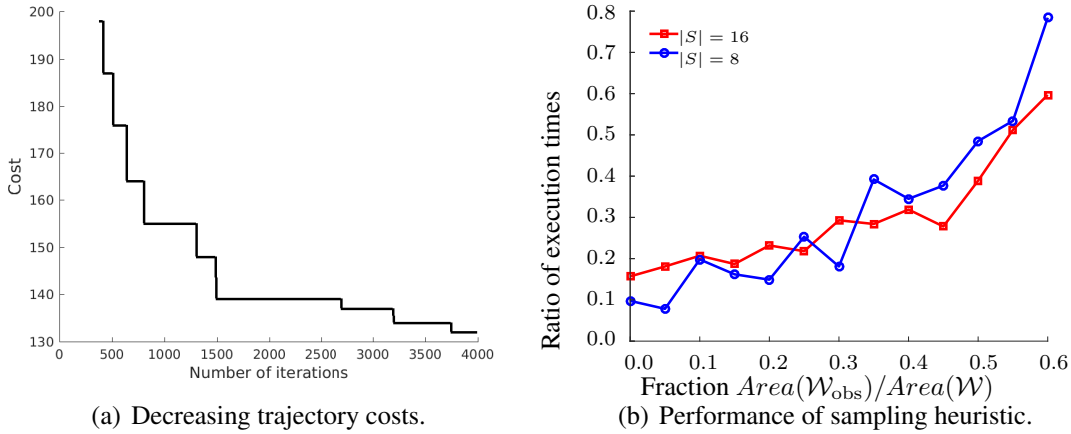


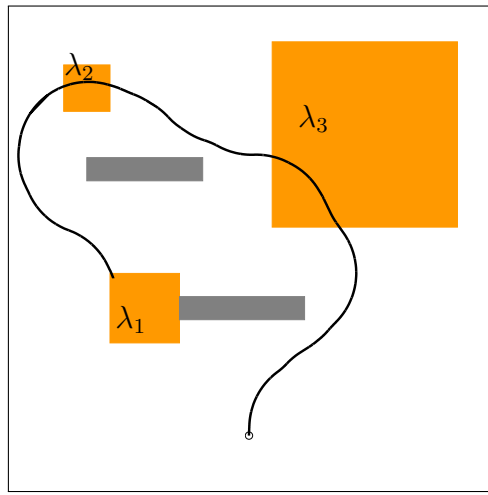
Figure 5.6: Results for Example 1.

Example 2. An example with a more complex LTL specification is illustrated in Fig. 5.8. Here, the workspace is a square of 200×200 units, and the vehicle’s minimum radius of turn is $\rho = 20$ units. The initial state is $(100, 20, \pi/2)$, indicated by a small circle. The task assigned to the vehicle is to visit regions λ_1, λ_2 and λ_3 , and to persistently visit λ_4, λ_5 and λ_6 infinitely often, and to always avoid the regions λ_7 and λ_8 . The LTL specification for this task is

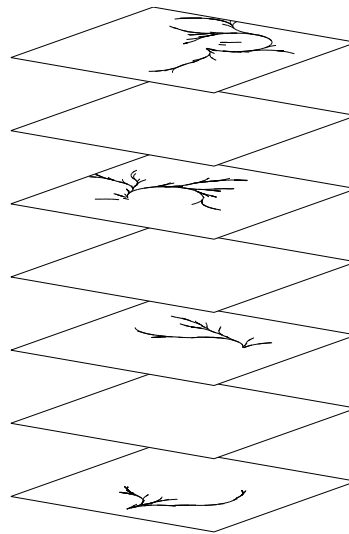
$$\phi_2 = \diamond \lambda_1 \wedge \diamond \lambda_2 \wedge \diamond \lambda_3 \wedge \square (\diamond \lambda_4 \wedge \diamond \lambda_5 \wedge \diamond \lambda_6) \wedge \square (\neg \lambda_7 \wedge \neg \lambda_8).$$

This specification involves the $\square \diamond$ (always eventually) operator to specify persistent surveillance, and lies outside the class of co-safe LTL specifications.

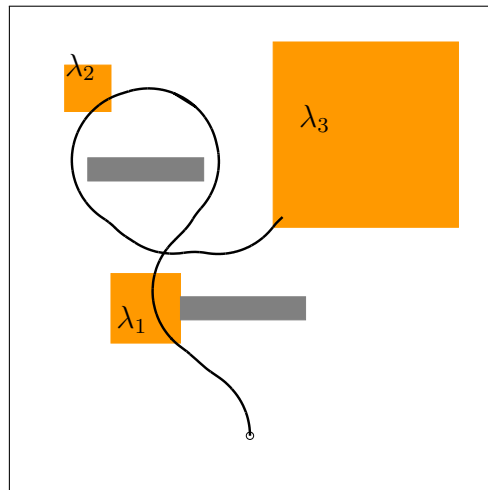
A reference trajectory for the prefix and one loop of the suffix of the path in satisfying this LTL specification is shown in Fig. 5.8. Notice the “loop” through regions λ_6, λ_4 , and λ_5 , to satisfy the persistent surveillance requirement. This first feasible solution was obtained after 17052 iterations, with an execution time of 89.22 s. This execution time compares highly favorably against the reported time of 210 s in the state-of-the-art [126]. Therein, a similar 3-state model with a simpler specification (i.e., fewer states in the Büchi automaton) is considered. The specification considered in [126] is to reach a target set while avoiding obstacles, which is simpler even compared Example 1 considered above, and far simpler than the specification here in Example 2.



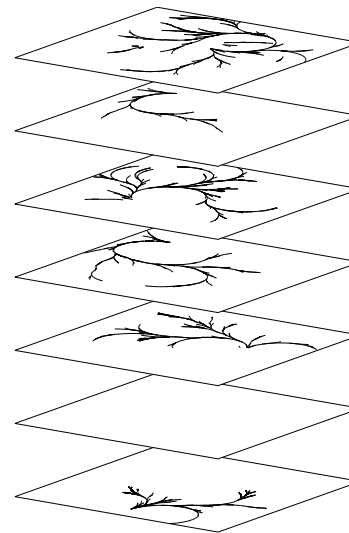
(a)



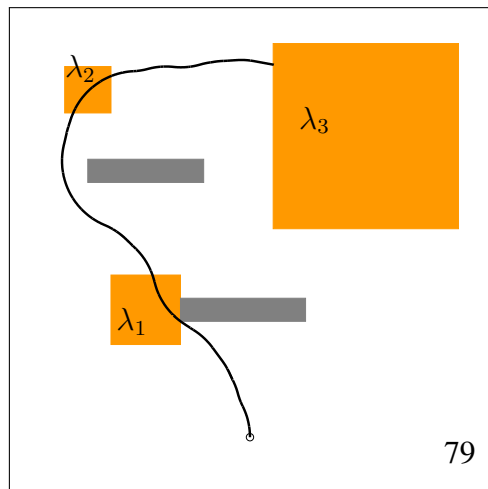
(b)



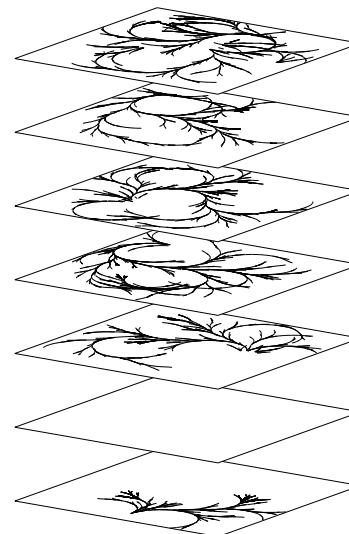
(c)



(d)



(e)



(f)

Figure 5.7: Simulation results for specification ϕ_1 in Example 1 with the fixed wing aircraft

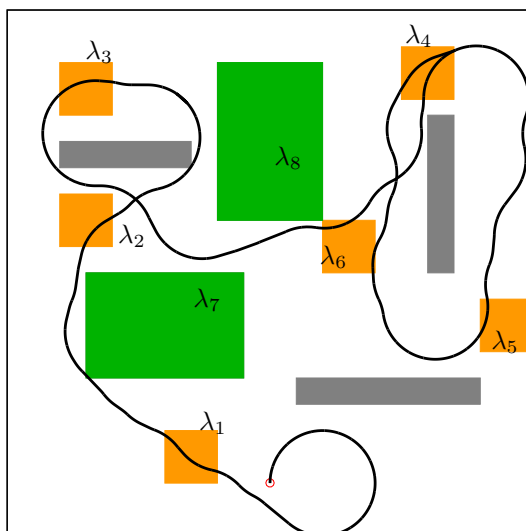


Figure 5.8: Simulation results for specification ϕ_2 in Example 2 with the fixed wing aircraft model (5.1).

Example 3. LTL formulae can express specifications of behavior *conditional* on the environment. For example, the following specification $\phi_3 := \diamond(\lambda_1 \wedge (Target \rightarrow \diamond\lambda_2)) \wedge \diamond\Box\lambda_3$ encodes a requirement of visiting region λ_0 , then visiting region λ_2 only if a target is found in region λ_1 , and returning to base λ_3 in either case. Such behavior may be required, for instance, to relay target surveillance data to a remote data station, cf. [127].

Here, the mechanism of detecting the target is irrelevant. For the scope of this work, we are concerned with the behavior of the vehicle in either case (i.e. target present / not present). To this end, the proposed algorithm is implemented for the workspace shown in Fig. 5.9. The detection of the target is hard-coded by a boolean variable within the software implementation of the proposed algorithm. In practice this variable can be replaced by a boolean estimate of the target’s presence generated by appropriate sensing hardware and software. As seen in Fig. 5.9(a), when the target is present, the reference trajectory passes through region λ_1 , otherwise it simply returns to base λ_2 (Fig. 5.9(b)). It is easy to include a persistent surveillance requirement, as in the preceding example with specification ϕ_2 .

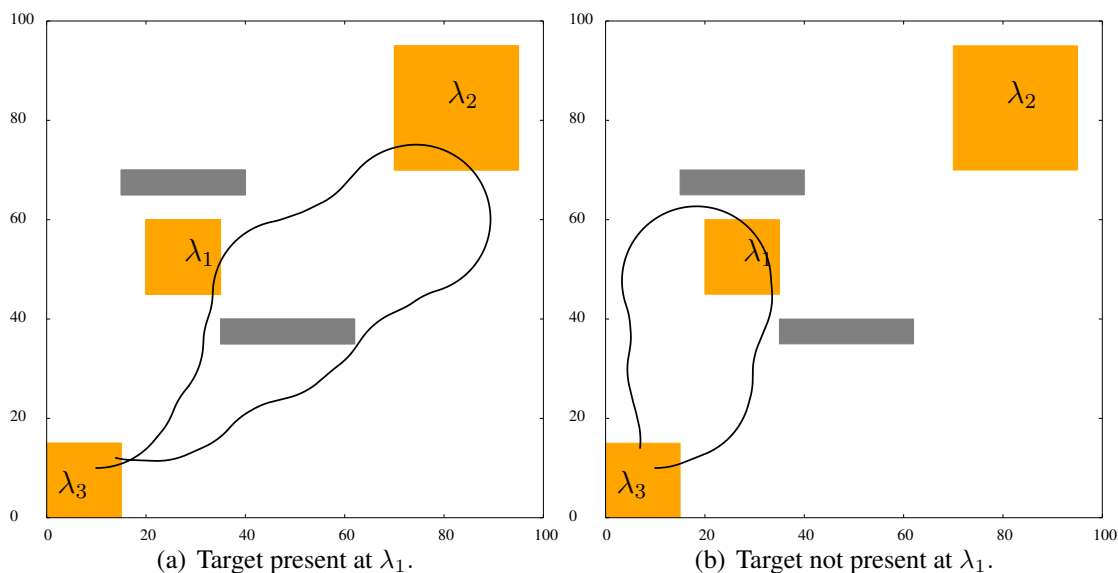


Figure 5.9: Simulation results for specification ϕ_3 illustrating trajectories conditional on properties of the environment.

5.4.3 Example with Quadrotor Aircraft Dynamical Model

This example of trajectory generation for specification ϕ_3 , implemented with the quadrotor dynamical model (5.2) and (5.3). The results are shown in Fig. 5.10, and show behavior similar to that for the fixed wing aircraft model shown in Fig. 5.9. In Fig. 5.10, the cubic shaded region close to $(0, 0, 0)$ is the base λ_3 , the cubic shaded region close to the center is λ_1 and the cubic shaded region near the top center is λ_2 . The tall shaded regions near the center are obstacles. Some parts of the trajectories are hidden by the shaded regions.

Figure 5.11 shows the performance advantage due to the sampling heuristic. As in Figs. 5.5 and 5.6(b), the horizontal axis is the ratio of the area occupied by obstacles to the area of the entire workspace, whereas the vertical axis is the ratio of the execution times of the proposed algorithm to find a first feasible solution with and without the proposed sampling heuristic. Each data point in Fig. 5.11 is an average value over 20 simulation instances, each with different obstacle locations and sizes.

The striking similarity of these three figures indicates that the strong performance ad-

5.4. NUMERICAL SIMULATION EXAMPLES

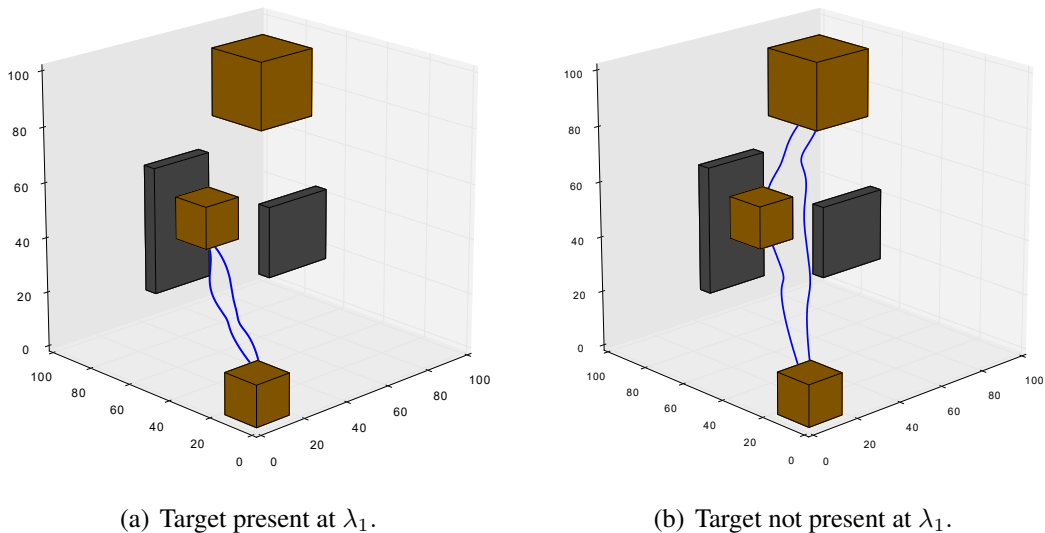


Figure 5.10: Simulation results for specification ϕ_3 illustrating quadrotor trajectories conditional on properties of the environment.

vantage gained due to the proposed sampling heuristic is independent of the complexity of the LTL specification (number of states in the Büchi automaton), and independent of the vehicle kinematical or dynamical model considered.

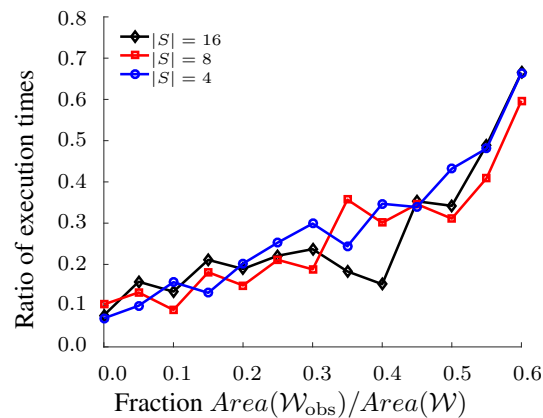


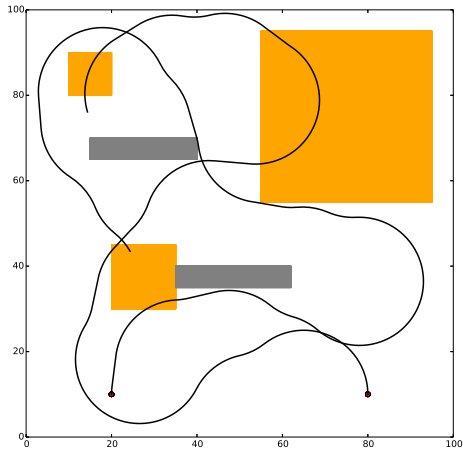
Figure 5.11: Performance of sampling heuristic for specification ϕ_3 .

5.5 Extension to Multiple Vehicles

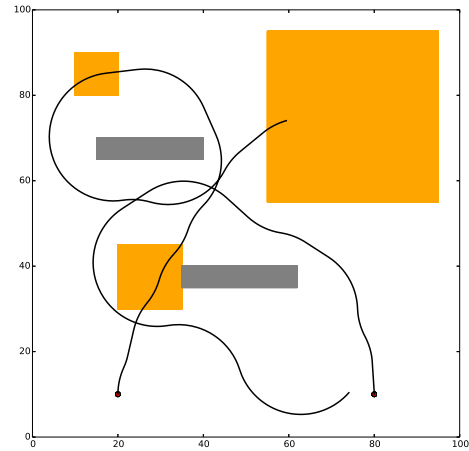
It is convenient to extend the algorithm we discussed in /refsec-algorithms to multiple vehicles, similar to the team state in 3.2: *team state* $\mu = (x_1, \dots, x_{N^V}) \in (\mathcal{D})^{N^V}$ is defined as the N^V -tuple of vertices in \mathcal{D} indicating the state of each vehicle. And then apply the algorithm on *product space* $T := (\mathcal{D})^{N^V} \times S$, the process of the algorithm remains the same.

Simulation result are shown in Fig. 5.12 using fixed wing aircraft kinematical model (5.1), the simulation task is the same task in refssec-example1 but assigned to two vehicles to visit regions $\lambda_1, \lambda_2, \lambda_3$ with no imposition on the order of visits, i.e., the LTL specification is $\phi_1 = \diamond\lambda_1 \wedge \diamond\lambda_2 \wedge \diamond\lambda_3$.

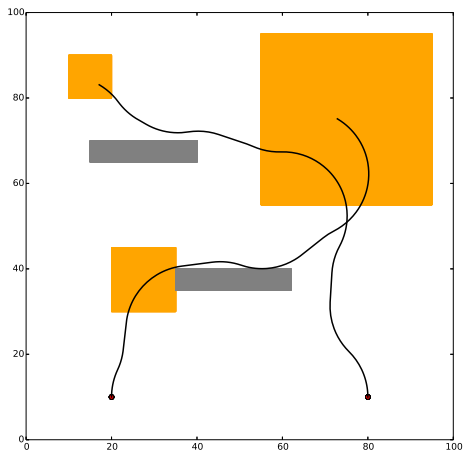
5.5. EXTENSION TO MULTIPLE VEHICLES



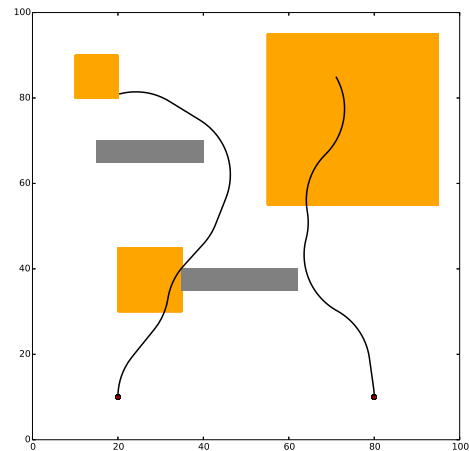
(a) First feasible solution.



(b) Solution after 800 iterations.



(c) Solution after 1500 iterations.



(d) Solution after 3500 iterations.

Figure 5.12: Simulation result for specification $\phi_1 = \diamond\lambda_1 \wedge \diamond\lambda_2 \wedge \diamond\lambda_3$.

Chapter 6

Conclusions and Directions of Future Work

In this thesis, we discussed a new motion-planning technique for autonomous vehicles subject to linear temporal logic specifications.

We proposed a graph based motion-planning algorithm relies on workspace partitioning, which involves partitioning in a space of smaller dimension than the *state* space, as is often done in the literature. The proposed technique relied on the so-called lifted graph, and on appropriate assignments of edge transition costs in the lifted graph. The relationship of these edge transition costs with certain forward- and backward reachable sets of the vehicle model was discussed, and the offline preprocessing of the computations of these sets was emphasized. The proposed approach is applicable more generally to differentially flat nonlinear systems, especially when control input constraints can be mapped to constraints on the flat output-space trajectories. The control constraints for the vehicle model considered in this paper were mapped to curvature constraints on the admissible workspace (output space) trajectories. Therefore, analysis of forward- and backward reachable sets with curvature-bounded curves was directly applied for finding lifted graph edge transition costs for this vehicle model.

We presented a multi-vehicle motion-planning algorithm for satisfying global LTL specifications, such that the resultant paths are traversable with admissible trajectories of the vehicle dynamical model. An incremental motion-planning algorithm was proposed, and illustrative numerical simulation results were presented.

We discussed an incremental algorithm for hierarchical motion-planning based on the previously developed H -cost motion-planning technique. The H -cost technique has been shown to be useful in the tight integration of the solutions of a high-level discrete task-

planning problem with a low-level continuous trajectory generation problem. Whereas it is beneficial to use high values of the parameter H , the complexity of the H -cost optimal path problem increases exponentially with H . The proposed algorithm alleviates this high computational cost by making available a *feasible* solution at intermediate iterations. We proved that the proposed algorithm is guaranteed to converge to an optimal solution given enough computation time (i.e. after a sufficiently large number of iterations). Furthermore, the cost of the solutions available in intermediate iterations of the proposed algorithm is always nonincreasing except, possibly, at a finite number of special iterations (i.e. when H is incremented).

We proposed a randomized sampling based algorithm for satisfying linear temporal logic (LTL) specifications similar to the well-known rapidly-exploring random tree (RRT*) algorithm. We proposed a sampling heuristic that provides significant reductions in execution time for finding feasible solutions. This sampling heuristic preserves the guarantees of probabilistic completeness and asymptotic optimality inherited from RRT*. We demonstrated the computational advantages of this sampling heuristic through a series of numerical simulations. In contrast to other randomized sampling-based approaches reported in the literature, the proposed algorithm is capable of finding infinite trajectories in prefix-suffix form to satisfy LTL specifications, i.e., it is not restricted to the class of co-safe LTL specifications.

Following are the possible future extensions of the proposed motion planning framework.

Motion-Planning with Uncertainties In this thesis, we assumed perfect knowledge of the environment and the vehicle. In practice, the environment is uncertain, the environment may be modeled by a Markov decision process (MDP), we can also use the idea of lifted graph discussed in 2.2 to incorporate vehicle dynamical constraints by modeling the motion planning problem by "lifted MDP".

Motion-Planning Algorithm on Multiple Vehicles We discussed a centralized motion-planning algorithm in chapter 3, future work includes removing the previously discussed

simplifying assumptions, including temporal synchronization, and the extension to LTL specifications involving the next operator and incorporation of a collision avoidance constraint among vehicles. Also we will address decentralization of the framework, one idea is to decompose the task that described by LTL formula by consensus-based bundle algorithm and then assign subtasks to each vehicle.

Bibliography

- [1] A. Pnueli, “The temporal logic of programs,” in *18th Annual Symposium on Foundations of Computer Science*, Providence, RI, USA, October 31 - November 2 1977, pp. 46–57.
- [2] V. S. Alagar and K. Periasamy, *Specification of Software Systems*, 2nd ed. London, UK: Springer-Verlag, 2011.
- [3] C. Baier and J.-P. Katoen, *Principles of Model Checking*. Cambridge, MA, USA: The MIT Press, 2008.
- [4] C. Belta, V. Isler, and G. J. Pappas, “Discrete abstractions for robot motion planning and control in polygonal environments,” *IEEE Transactions on Robotics*, vol. 21, no. 5, pp. 864 – 874, October 2005.
- [5] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, “Hybrid controllers for path planning: A temporal logic approach,” in *Proceedings of the 44th IEEE Conference on Decision and Control*, Seville, Spain, 12 – 15 Dec. 2005, pp. 4885 – 4890.
- [6] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. J. Pappas, “Symbolic planning and control of robot motion,” *IEEE Robotics and Automation Magazine*, pp. 61 – 70, March 2007.
- [7] D. J. Bertsimas and D. Simichi-Levi, “A new generation of vehicle routing research: Robust algorithms, addressing uncertainty,” *Operations Research*, vol. 44, no. 2, pp. 286–304, 1996.
- [8] V. Pillac, M. Gendreau, C. Gueret, and M. A. L., “A review of dynamic vehicle routing problems,” *European Journal of Operational Research*, vol. 225, pp. 1–35, 2013.

BIBLIOGRAPHY

- [9] C. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Mineola, NY, USA: Dover Publications, Inc., 1998.
- [10] D. J. Bertsimas and G. Van Ryzin, “A stochastic and dynamic vehicle routing problem in the euclidean plane,” *Operations Research*, vol. 39, no. 4, pp. 601–615, 1991.
- [11] S. D. Bopardikar, S. L. Smith, F. Bullo, and J. P. Hespanha, “Dynamic vehicle routing for translating demands: Stability analysis and receding-horizon policies,” *IEEE Transactions on Automatic Control*, vol. 55, pp. 2554–2569, 2010.
- [12] X. Ma and D. A. Castanon, “Receding horizon planning for dubins traveling salesman problems,” in *Proceedings of the 45th IEEE Conference on Decision & Control*, San Diego, CA, USA, Dec. 13–15 2006, pp. 5453–5458.
- [13] K. Savla, E. Frazzoli, and F. Bullo, “Traveling salesperson problems for the dubins vehicle,” *IEEE Transactions on Automatic Control*, vol. 53, pp. 1378–1391, 2008.
- [14] J. Cortés, S. Martinez, T. Karatas, and F. Bullo, “Coverage control for mobile sensing networks,” *IEEE Transactions on Robotics and*, vol. 20, no. 2, pp. 243–255, 2004.
- [15] S. Martinez, J. Cortes, and F. Bullo, “Motion coordination with distributed information,” *IEEE Control Systems Magazine*, pp. 75–88, August 2007.
- [16] F. Bullo, J. Cortés, and S. Martinez, *Distributed Control of Robotic Networks: A Mathematical Approach to Motion Coordination Algorithms*. Princeton University Press, 2009.
- [17] Z. Shiller and Y.-R. Gwo, “Dynamic motion planning of autonomous vehicles,” *IEEE Transactions on Robotics and Automation*, vol. 7, no. 2, pp. 241–249, 1991.
- [18] D. Zhu and J.-C. Latombe, “New heuristic algorithms for efficient hierarchical path planning,” *IEEE Transactions on Robotics and Automation*, vol. 7, no. 1, pp. 9–20, 1991.

BIBLIOGRAPHY

- [19] S. R. Cunha, A. C. de Matos, and F. L. Pereira, “An automatic path planning system for autonomous robotic vehicles,” in *Proceedings of the IECON '93 International Conference on Industrial Electronics, Control, and Instrumentation*, Maui, HI, USA, November 1993, pp. 1442–1447.
- [20] J.-P. Laumond, M. Taix, P. Jacobs, and R. M. Murray, “A motion planner for nonholonomic mobile robots,” *IEEE Transactions on Robotics and Automation*, vol. 10, no. 5, pp. 577–593, 1994.
- [21] M. Cherif, “Kinodynamic motion planning for all-terrain wheeled vehicles,” in *Proceedings of the 1999 IEEE International Conference on Robotics and Automation*, Detroit, MI., May 1999, pp. 317 – 322.
- [22] D. Coombs, K. Murphy, A. Lacaze, and S. Legowik, “Driving autonomously offroad up to 35 km/h,” in *Proceedings of the 2000 International Vehicles Conference*, 2000.
- [23] A. Rosiglioni and M. Simina, “Kinodynamic motion planning,” in *Proceedings of the IEEE Conference on Systems, Man, and Cybernetics*, vol. 3, 2003, pp. 2243–2248.
- [24] B. Mettler and E. Bachelder, “Combining on- and offline optimization techniques for efficient autonomous vehicle’s trajectory planning,” in *Collection of Technical Papers - AIAA Guidance, Navigation, and Control Conference 2005*, ser. 499–511, vol. 1, 2005.
- [25] J.-C. Latombe, *Robot Motion Planning*. Kluwer Academic Publishers, 1991.
- [26] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. The MIT Press, 2005.
- [27] R. A. Brooks and T. Lozano-Pérez, “A subdivision algorithm in configuration space for findpath with rotation,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-15, no. 2, pp. 224–233, Mar–Apr 1985.

BIBLIOGRAPHY

- [28] M. de Berg, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*. Berlin: Springer, 1997.
- [29] H. Samet, “The quadtree and related hierarchical data structures,” *Computing Surveys*, vol. 16, no. 2, pp. 187–260, June 1984.
- [30] S. Kambhampati and L. S. Davis, “Multiresolution path planning for mobile robots,” *IEEE Journal of Robotics and Automation*, vol. RA-2, no. 3, pp. 135–45, September 1986.
- [31] H. Noborio, T. Naniwa, and S. Arimoto, “A quadtree-based path planning algorithm for a mobile robot,” *Journal of Robotic Systems*, vol. 7, no. 4, pp. 555–74, 1990.
- [32] J. Y. Hwang, J. S. Kim, S. S. Lim, and K. H. Park, “A fast path planning by path graph optimization,” *IEEE Transactions on Systems, Man, and Cybernetics– Part A: Systems and Humans*, vol. 33, no. 1, pp. 121–127, January 2003.
- [33] S. Behnke, “Local multiresolution path planning,” *Lecture Notes in Artificial Intelligence*, vol. 3020, pp. 332–43, 2004.
- [34] P. Tsiotras and E. Bakolas, “A hierarchical on-line path planning scheme using wavelets,” in *Proceedings of the European Control Conference*, Kos, Greece, July 2–5 2007, pp. 2806–2812.
- [35] A. E. Bryson and Y.-C. Ho, *Applied Optimal Control*. New York, NY, USA: Taylor & Francis, 1975.
- [36] L. Cesari, *Optimization - Theory and Applications*. New York, NY, USA: Springer-Verlag, 1983.
- [37] M. Athans and P. L. Falb, *Optimal Control*. Dover Publications, Inc., 2007.
- [38] J. M. Longuski, J. J. Guzman, and J. E. Prussing, *Optimal Control with Aerospace Applications*. New York, Ny, USA: Springer, 2014.

- [39] G. M. Siouris, *Missile Guidance and Control Systems*. New York, NY: Springer, 2003.
- [40] X.-N. Bui, J.-D. Boissonnat, P. Souères, and J.-P. Laumond, “Shortest path synthesis for dubins non-holonomic robot,” in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, San Diego, CA, USA, May 1994, pp. 2–7.
- [41] H. J. Sussmann, “Shortest 3-dimensional paths with a prescribed curvature bound,” in *Proceedings of the 34th IEEE Conference on Decision & Control*, New Orleans, LA, 1995, pp. 3306–3312.
- [42] J.-D. Boissonnat and X.-N. Bui, “Accessibility region for a car that only moves forwards along optimal paths,” Institut National de Recherche en Informatique et Automatique (INRIA), Tech. Rep., 1994.
- [43] T. G. McGee and J. K. Hedrick, “Optimal path planning with a kinematic airplane model,” *Journal of Guidance, Control, and Dynamics*, vol. 30, no. 2, pp. 629–633, 2007.
- [44] P. K. A. Menon and E. Kim, “Optimal trajectory synthesis for terrain-following flight,” *Journal of Guidance, Control, and Dynamics*, vol. 14, no. 4, pp. 807–813, 1991.
- [45] E. Bakolas and P. Tsotras, “Optimal synthesis of the Zermelo-Markov-Dubins problem in a constant drift field,” *Journal of Optimization Theory and Applications*, vol. 156, no. 2, pp. 469–492, 2013.
- [46] J. T. Betts, “Survey of numerical methods for trajectory optimization,” *Journal of Guidance, Control, and Dynamics*, vol. 21, no. 2, pp. 193–204, 1998.
- [47] I. M. Ross and F. Fahroo, *Legendre Pseudospectral Approximations in Optimal Control Problems*, ser. Lecture Notes in Control and Information Sciences. Berlin, Germany: Springer-Verlag, 2003, vol. 295, pp. 327–342.

- [48] A. Farooq and D. J. N. Limebeer, "Trajectory optimization for air-to-surface missiles with imaging radars," *Journal of Guidance, Control, and Dynamics*, vol. 25, no. 5, pp. 876–887, 2002.
- [49] B. Sridhar, H. K. Ng, and N. Y. Chen, "Aircraft trajectory optimization and contrails avoidance in the presence of winds," *Journal of Guidance, Control, and Dynamics*, vol. 34, pp. 1577–1583, 2011.
- [50] Y. Zhao and P. Tsiotras, "Analysis of energy-optimal aircraft landing operation trajectories," *Journal of Guidance, Control, and Dynamics*, vol. 36, no. 3, pp. 833–845, 2013.
- [51] P. Williams, "Three-dimensional aircraft terrain-following via real-time optimal control," *Journal of Guidance, Control, and Dynamics*, vol. 30, pp. 1201–1205, 2007.
- [52] Z. Shiller and H.-H. Lu, "Computation of path constrained time-optimal motions with dynamic singularities," *ASME Journal of Dynamic Systems, Measurement, and Control*, vol. 114, pp. 34–40, 1992.
- [53] Z. Shiller, "On singular time-optimal control along specified paths," *IEEE Transactions on Robotics and Automation*, vol. 10, no. 4, pp. 561–566, 1994.
- [54] M. Lepetič, G. Klancčar, I. Škrjanc, D. Matko, and B. Potočnik, "Time optimal path planning considering acceleration limits," *Robotics and Autonomous Systems*, vol. 45, pp. 199–210, 2003.
- [55] E. Velenis and P. Tsiotras, "Minimum-time travel for a vehicle with acceleration limits: Theoretical analysis and receding horizon implementation," *Journal of Optimization Theory and Applications*, vol. 138, no. 2, pp. 275–296, 2008.
- [56] B. D. O. Anderson and J. B. Moore, *Optimal Control: Linear Quadratic Methods*. Englewood Cliffs, NJ, USA: Prentice Hall, 1990.

BIBLIOGRAPHY

- [57] J. N. Tsitsiklis, “Efficient algorithms for global optimal trajectories,” *IEEE Transactions on Automatic Control*, vol. 40, no. 9, pp. 1528–1538, 1995.
- [58] D. P. Bertsekas, *Dynamic Programming and Optimal Control*. Athena Scientific, Belmont, MA, 2000.
- [59] E. Rippel, A. Bar-Gill, and N. Shimkin, “Fast graph-search algorithms for general aviation flight trajectory generation,” *Journal of Guidance, Control, and Dynamics*, vol. 28, no. 4, pp. 801–811, July-August 2005.
- [60] N. Faiz and S. K. Agrawal, “Trajectory planning of robots with dynamics and inequalities,” in *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, San Francisco, CA, April 2000, pp. 3976–3981.
- [61] T. Schouwenaars, B. Mettler, E. Feron, and J. How, “Hybrid model for trajectory planning of agile autonomous vehicles,” *Journal of Aerospace Computing, Information, and Communication*, vol. 1, pp. 629–651, 2004.
- [62] S. Sundar and Z. Shiller, “Optimal obstacle avoidance based on Hamilton-Jacobi-Bellman equation,” *IEEE Transactions on Robotics and Automation*, vol. 13, no. 2, pp. 305–310, April 1997.
- [63] L. E. Dubins, “On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents,” *American Journal of mathematics*, vol. 79, no. 3, pp. 497–516, 1957.
- [64] J. A. Reeds and L. A. Shepp, “Optimal paths for a car that goes both forwards and backwards,” *Pacific Journal of Mathematics*, vol. 145, no. 2, pp. 367–393, 1990.
- [65] J. Reif and H. Wang, “The complexity of the two dimensional curvature-constrained shortest-path problem,” in *Proceedings of the 3rd Workshop on the Algorithmic Foundations of Robotics*, 1998, pp. 49–58.

BIBLIOGRAPHY

- [66] B. Donald, P. Xavier, J. Canny, and J. Reif, “Kinodynamic motion planning,” *Journal of the Association for Computing Machinery*, vol. 40, no. 5, pp. 1048–1066, November 1993.
- [67] L. E. Kavraki and J.-C. Latombe, “Randomized preprocessing of configuration space for fast path planning,” Dept. Computer Science, Stanford University, Stanford, CA, Tech. Rep. STAN-CS-93-1490, 1993.
- [68] P. Švestka, “A probabilistic approach to motion planning for car-like robots,” Dept. Computer Science, Utrecht University, Utrecht, The Netherlands, Tech. Rep. RUU-CS-1993-18, 1993.
- [69] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 4, pp. 566–580, 1996.
- [70] N. M. Amato and Y. Wu, “A randomized roadmap method for path and manipulation planning,” in *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, vol. 1. IEEE, 1996, pp. 113–120.
- [71] S. M. LaValle and J. J. Kuffner, “Randomized kinodynamic planning,” *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.
- [72] S. M. LaValle and J. J. Kuffner, Jr., “Randomized kinodynamic planning,” *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, May 2001.
- [73] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, “Randomized kinodynamic motion planning with moving obstacles,” *International Journal of Robotics Research*, vol. 21, no. 3, pp. 233–255, March 2002.
- [74] E. Frazzoli, M. A. Dahleh, and E. Feron, “Real-time motion planning for agile autonomous vehicles,” *Journal of Guidance, Control, and Dynamics*, vol. 25, no. 1, pp. 116–129, 2002.

- [75] E. Plaku, L. E. Kavraki, and M. Y. Vardi, “Motion planning with dynamics by a synergistic combination of layers of planning,” *IEEE Transactions on Robotics*, vol. 26, no. 3, pp. 469–482, 2010.
- [76] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *International Journal of Robotics Research*, vol. 30, pp. 846–894, 2011.
- [77] L. Kavraki, P. Svestka, and M. H. Overmars, *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*. Unknown Publisher, 1994, vol. 1994.
- [78] J. Barraquand, L. Kavraki, R. Motwani, J.-C. Latombe, T.-Y. Li, and P. Raghavan, “A random sampling scheme for path planning,” in *Robotics Research*. Springer, 1996, pp. 249–264.
- [79] D. Hsu, J.-C. Latombe, and H. Kurniawati, “On the probabilistic foundations of probabilistic roadmap planning,” *The International Journal of Robotics Research*, vol. 25, no. 7, pp. 627–643, 2006.
- [80] M. Antoniotti and B. Mishra, “Discrete event models temporal logic supervisory controller: Automatic synthesis of locomotion controllers,” in *Proceedings of the 1995 International Conference on Robotics and Automation*, vol. 2. IEEE, 1995, pp. 1441–1446.
- [81] S. G. Loizou and K. J. Kyriakopoulos, “Automatic synthesis of multi-agent motion tasks based on ltl specifications,” in *Proceedings of the 2004 43rd IEEE Conference on Decision and Control*, vol. 1. IEEE, 2004, pp. 153–158.
- [82] M. Kloetzer and C. Belta, “A fully automated framework for control of linear systems from ltl specifications,” in *International Workshop on Hybrid Systems: Computation and Control*, vol. 3927. Springer, 2006, pp. 333–347.
- [83] M. M. Quottrup, T. Bak, and R. Zamanabadi, “Multi-robot planning: A timed automata approach,” in *Proceedings of the 2004 IEEE International Conference on Robotics and Automation*, vol. 5. IEEE, 2004, pp. 4417–4422.

- [84] A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, and D. Rus, “Optimality and robustness in multi-robot path planning with temporal logic constraints,” *International Journal of Robotics Research*, vol. 32, no. 8, pp. 889–911, 2013.
- [85] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas, “Discrete abstractions of hybrid systems,” *Proceedings of the IEEE*, vol. 88, no. 7, pp. 971–984, July 2000.
- [86] P. Tabuada, *Verification and Control of Hybrid Systems: A Symbolic Approach*. Springer, 2008.
- [87] M. Kloetzer and C. Belta, “A fully automated framework for control of linear systems from temporal logic specifications,” *IEEE Transactions on Automatic Control*, vol. 53, no. 1, pp. 287–297, February 2008.
- [88] P. Wolper, M. Vardi, and A. Sistla, “Reasoning about infinite computations,” in *Proceedings of the 24th Symposium on Foundations of Computer Science*, Tucson, AZ, 1983, pp. 185–194.
- [89] P. Wolper, “Constructing automata from temporal logic formulas: A tutorial,” in *Formal Methods Performance Analysis: First EEF/Euro Summer School on Trends in Computer Science*. New York, NY: Springer-Verlag, 2001.
- [90] G. Holzmann, “The model checker SPIN,” *IEEE Transactions on Software Engineering*, vol. 23, no. 5, pp. 279–295, 1997.
- [91] P. Gastin and D. Oddoux, “Fast LTL to Büchi automata translation,” in *Proceedings of the 13th Conference on Computer Aided Verification (CAV’ 01)*, ser. Lecture Notes in Computer Science, H. C. G. Berry and A. Finkel, Eds., vol. 2102. New York: Springer-Verlag, 2001, pp. 53–65.
- [92] X. C. Ding, M. Lazar, and C. Belta, “LTL receding horizon control for finite deterministic systems,” *Automatica*, vol. 50, pp. 399–408, 2014.

- [93] J. A. DeCastro and H. Kress-Gazit, “Synthesis of nonlinear continuous controllers for verifiably correct high-level, reactive behaviors,” *International Journal of Robotics Research*, vol. 34, no. 3, pp. 378–394, 2015.
- [94] M. Kloetzer and C. Belta, “Temporal logic planning and control of robotic swarms by hierarchical abstractions,” *IEEE Transaction on Robotics*, vol. 23, no. 2, pp. 320–330, 2007.
- [95] Y. Yordanov, J. Tůmova, I. Černá, J. Barnat, and C. Belta, “Temporal logic control of discrete-time piecewise affine systems,” *IEEE Transactions on Automatic Control*, vol. 57, no. 6, pp. 1491–1505, 2012.
- [96] M. Zamani, G. Pola, M. Mazo, and P. Tabuada, “Symbolic models for nonlinear control systems without stability assumptions,” *IEEE Transactions on Automatic Control*, vol. 57, no. 7, pp. 1804–1809, 2012.
- [97] E. M. Wolff, U. Topcu, and R. M. Murray, “Optimization-based control of nonlinear systems with linear temporal logic specifications,” in *Proceedings of the 2014 International Conference on Robotics and Automation*, Hong Kong, China, May 31 – June 7 2014, pp. 5319–5325.
- [98] H. Kress-Gazit, D. C. Conner, H. Choset, A. A. Rizzi, and G. J. Pappas, “Courteous cars: Decentralized multiagent traffic coordination,” *IEEE Robotics & Automation Magazine*, vol. 15, pp. 30–38, 2008.
- [99] S. Coogan and M. Arcak, “Efficient finite abstraction of mixed monotone systems,” in *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, Seattle, WA, USA, April 14 – 16 2015, pp. 58–67.
- [100] M. Guo and D. V. Dimarogonas, “Reconfiguration in motion-planning of single- and multi-agent systems under infeasible local LTL specifications.” in *Proceedings of the 52nd IEEE Conference on Decision & Control*, Florence, Italy, December 2013.

BIBLIOGRAPHY

- [101] —, “Multi-agent plan reconfiguration under local LTL specifications,” *International Journal of Robotics Research*, vol. 34, no. 2, pp. 218–235, 2015.
- [102] R. V. Cowlagi and P. Tsiotras, “Hierarchical motion planning with dynamical feasibility guarantees for mobile robotic vehicles,” *IEEE Transactions on Robotics*, vol. 28, no. 2, pp. 379 – 395, 2012.
- [103] A. Stentz, “The focussed D* algorithm for real-time replanning,” in *Proceedings of the International Joint Conference on Artificial Intelligence*, vol. 95, 1995, pp. 1652–1659.
- [104] T. Wongpiromsarn, V. G. Rao, and R. D. D’Andrea, “Two approaches to dynamic refinement in hierarchical motion planning,” in *Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit*, San Francisco, CA, USA, Aug 15–18 2005.
- [105] P. Tabuada and G. J. Pappas, “Model checking LTL over controllable linear systems is decidable,” in *Hybrid Systems: Computation & Control*, ser. LNCS 2623, O. Maler and A. Pnueli, Eds. Berlin: Springer-Verlag, 2003, pp. 498–513.
- [106] C. I. Vasile and C. Belta, “Sampling-based temporal logic path planning,” in *Proceedings of the 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013, pp. 4817–4822.
- [107] V. Varricchio, P. Chaudhari, and E. Frazzoli, “Sampling-based algorithms for optimal motion planning using process algebra specifications,” in *Proceedings of the 2014 IEEE International Conference on Robotics and Automation*. IEEE, 2014, pp. 5326–5332.
- [108] D. P. Bertsekas and I. B. Rhodes, “On the minimax reachability of target sets and target tubes,” *Automatica*, vol. 7, pp. 233–247, 1971.
- [109] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to automata theory, languages, and computation*, 2nd ed. Boston, MA, USA: Addison-Wesley, 2001.

BIBLIOGRAPHY

- [110] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. MIT Press, 2001.
- [111] J. Cichon, A. Czubak, and A. Jasinski, “Minimal büchi automata for certain classes of ltl formulas,” in *2009 Fourth International Conference on Dependability of Computer Systems*. IEEE, 2009, pp. 17–24.
- [112] R. V. Cowlagi and P. Tsiotras, “Curvature-bounded traversability analysis for motion planning of mobile robots,” *IEEE Transactions on Robotics*, vol. 30, no. 4, pp. 1011–1019, 2014.
- [113] R. V. Cowlagi, “Hierarchical motion planning for autonomous aerial and terrestrial vehicles,” Ph.D. dissertation, Georgia Institute of Technology, 2011.
- [114] E. J. Cockayne and G. W. C. Hall, “Plane motion of a particle subject to curvature constraints,” *SIAM Journal on Control*, vol. 13, no. 1, pp. 197–220, 1975.
- [115] S. Bereg and D. Kirkpatrick, “Curvature-bounded traversals of narrow corridors,” in *Proceedings of the Twenty-first Annual Symposium on Computational Geometry*, Pisa, Italy, 2005, pp. 278–287.
- [116] B. Dawes, D. Abrahams, and R. Rivers, “Boost C++ libraries,” Available online at <http://www.boost.org/>.
- [117] N. Edmonds, A. Breuer, D. Gregor, and A. Lumsdaine, “Single-source shortest paths with the parallel boost graph library,” in *The Ninth DIMACS Implementation Challenge: The Shortest Path Problem*, Piscataway, NJ, November 2006.
- [118] S. R. Lindemann, I. I. Hussein, and S. M. LaValle, “Real time feedback control for non-holonomic mobile robots with obstacles,” in *Proceedings of the 45th IEEE Conference on Decision and Control*, San Diego, CA, USA, December 2006, pp. 2406–2411.

BIBLIOGRAPHY

- [119] Z. Zhang and R. V. Cowlagi, “Incremental path repair in hierarchical motion-planning with dynamical feasibility guarantees for mobile robotic vehicles,” in *Proceedings of the 2015 European Control Conference*. IEEE, 2015, pp. 2366–2371.
- [120] S. Koenig, M. Likhachev, Y. Liu, and D. Furcy, “Incremental heuristic search in AI,” *Artificial Intelligence Magazine*, vol. 25, pp. 99–112, 2004.
- [121] D. Mellinger, N. Michael, and V. Kumar, “Trajectory generation and control for precise aggressive maneuvers with quadrotors,” *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 664–674, Apr. 2012.
- [122] M. Cutler and J. P. How, “Analysis and Control of a Variable-Pitch Quadrotor for Agile Flight,” *Journal of dynamic systems, measurement, and control*, vol. 137, no. 10, p. 101002, 1 Oct. 2015.
- [123] J. McMahan and E. Plaku, “Sampling-based tree search with discrete abstractions for motion planning with dynamics and temporal logic,” in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. IEEE, 2014, pp. 3726–3733.
- [124] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.
- [125] A. Duret-Lutz, A. Lewkowicz, A. Fauchille, T. Michaud, E. Renault, and L. Xu, “Spot 2.0 — a framework for LTL and ω -automata manipulation,” in *Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA’16)*, ser. Lecture Notes in Computer Science, vol. 9938. Springer, Oct. 2016, pp. 122–129.
- [126] M. Rungger and M. Zamani, “Scots: A tool for the synthesis of symbolic controllers,” in *Hybrid Systems Computation and Control*, 2016.

BIBLIOGRAPHY

- [127] D. J. Klein, S. Venkateswaran, J. T. Isaacs, J. Burman, T. Pham, J. a. Hespanha, and U. Madhow, “Localization with sparse acoustic sensor network using UAVs as information-seeking data mules,” *ACM Trans. Sen. Netw.*, vol. 9, no. 3, pp. 30:1–30:29, Jun. 2013.