Sampling-Based Motion Planning Algorithms for Dynamical Systems

by

Jeong hwan Jeon

S.M., Massachusetts Institute of Technology (2009)

B.S., Mechanical and Aerospace Eng., Seoul National University (2007)

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
September 2015

Author . . . .
Department of Aeronautics and Astronautics
August 20, 2015

Certified by. . . . .
Prof. Emilio Frazzoli
Professor of Aeronautics and Astronautics
Thesis Supervisor

Certified by. . . . .
Dr. Karl Iagnemma
Principal Research Scientist

Certified by. . . . .
Prof. Russell L. Tedrake
Associate Professor of EECS

Accepted by . . . .
Prof. Paulo C. Lozano
Associate Professor of Aeronautics and Astronautics
Chair, Graduate Program Committee

Sampling-Based Motion Planning Algorithms for Dynamical Systems

by

Jeong hwan Jeon

Submitted to the Department of Aeronautics and Astronautics
on August 20, 2015, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Dynamical systems bring further challenges to the problem of motion planning, by additionally complicating the computation of collision-free paths with collision-free dynamic motions. This dissertation proposes efficient approaches for the optimal sampling-based motion planning algorithms, with a strong emphasis on the accommodation of realistic dynamical systems as the subject of motion planning. The main contribution of the dissertation is twofold: advances in general framework for asymptotically-optimal sampling-based algorithms, and the development of fast algorithmic components for certain classes of dynamical systems.

The first part of the dissertation begins with key ideas from a number of recent sampling-based algorithms toward fast convergence rates. We reinterpret the ideas in the context of incremental algorithms, and integrate the key ingredients within the strict $\mathcal{O}(\log n)$ complexity per iteration, which we refer to as the enhanced RRT* algorithm. Subsequently, Goal-Rooted Feedback Motion Trees (GR-FMTs) are presented as an adaptation of sampling-based algorithms into the context of asymptotically-optimal feedback motion planning or replanning. Last but not least, we propose a loop of collective operations, or an efficient loop with cost-informed operations, which minimizes the exposure to the main challenges incurred by dynamical systems, i.e., steering problems or Two-Point Boundary Value Problems (TPBVPs).

The second main part of the dissertation directly deals with the steering problems for three categories of dynamical systems. First, we propose a numerical TPBVP method for a general class of dynamical systems, including time-optimal off-road vehicle maneuvers as the main example. Second, we propose a semi-analytic TPBVP approach for differentially flat systems or partially flat systems, by which the computation of vehicle maneuvers is expedited and the capability to handle extreme scenarios is greatly enhanced. Third, we propose an efficient TPBVP algorithm for controllable linear systems, based on the computation of small-sized linear or quadratic programming problems in a progressive and incremental manner.

Overall, the main contribution in this dissertation realizes the outcome of anytime algorithms for optimal motion planning problems. An initial solution is obtained within a small time, and the solution is further improved toward the optimal one. To our best knowledge from both simulation results and algorithm analyses, the proposed algorithms supposedly outperform or run at least as fast as other state-of-the-art sampling-based algorithms.

Thesis Supervisor: Prof. Emilio Frazzoli
Title: Professor of Aeronautics and Astronautics

Acknowledgments

First and foremost, I would like to thank my advisor Prof. Emilio Frazzoli for his generous support, diverse opportunities, inspirational advice, enthusiastic questioning, rigorous attitude to research, and patient guidance throughout the years of my graduate study at MIT. I have been extremely lucky to be a student under the supervision of such an insightful, untiring, and thorough researcher.

I am very grateful to other members in my thesis committee, Dr. Karl Iagnemma and Prof. Russ Tedrake, for their feedbacks and encouragement on both my thesis work and my career. I cannot form a better thesis committee with such perfect and balanced depths, breadths, and perspectives in my research topics. The readers of this dissertation, Prof. Sertac Karaman and Dr. Andrea Censi, cannot be thanked enough for endless inspiration and conversations, as a former colleague, a current colleague, and a faculty/staff member who have been the best source for learning how to grow up as a graduate student and a junior researcher by close observation.

I owe many years of unconditional support and excitement to my academic collaborators, labmates, staffs, and friends. All the valuable names cannot be enumerated in this limited space, but it is obvious that the years of my stay in Cambridge, MA could not be even imaginable without such wonderful supporters. Last but not least, my wife, parents, and sister have been the first and last resort and shelter during all good and bad times. Without them, I could not have driven myself toward the completion of my PhD degree.

This dissertation work was partially supported by the US Army Research Office, MURI grant W911NF-11-1-0046. In addition, my graduate study at MIT was generously supported in part by the STX Scholarship Foundation. Any opinions, findings, and conclusions or recommendations expressed in this dissertation are those of the author and do not necessarily reflect the views of the supporting organizations.

A substantial portion of the material in this dissertation is written with and without editing or rearrangement from the following papers.

© 2011 IEEE. Reprinted, with permission, from J. Jeon, S. Karaman., and

Contents

THIS PAGE INTENTIONALLY LEFT BLANK

List of Algorithms

List of Figures

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 1

Introduction

The three iterations of the DARPA Grand Challenge, culminating in the 2007 Urban Challenge, have successfully demonstrated autonomous robotic vehicles to the general public. Most participating teams adopted computation-intensive perception and planning frameworks, including motion planning algorithms [110, 81, 75]. In recent decades, an unprecedented level of autonomy has been progressively achieved by randomized algorithms such as Probabilistic RoadMaps (PRMs) [60, 18] and Rapidly-Exploring Random Trees (RRTs) [74, 65] that tackled previously daunting tasks in motion planning [119, 66, 4, 37, 105, 64, 67, 109]. Furthermore, robotic motions need to approach the physical limits of the system, especially when the robotic tasks require skilled capabilities to handle disasters, accidents, or cooperation [120, 50, 116].

We obtain robotic motions as answers to optimal motion planning problems with dynamical systems. Our formulation of the motion planning problem assumes deterministic models for the dynamical system.

1.1 Optimal Motion Planning

A typical definition of the *motion planning* [69, 26, 72] problem is, given an initial state, a goal set, an obstacle set, and a description of the system dynamics, to find time-parameterized trajectories of control inputs and resultant system states that safely drive the system from the initial state to the goal set while avoiding any collision with obstacles. This problem of navigating complex or cluttered environment is one of the fundamental topics in robotics [69], with applications including, but not limited to, autonomous driving [75], manipulation planning [5], logistics [109], rapid prototyping [24], and robotic surgery [107]. Outside the domain of robotics, the motion planning problem further finds its important applications, ranging from verification to computational biology [15, 59, 4].

An algorithm is said to be *complete*, if it surely answers to solvable problems in finite time. Otherwise, the complete algorithm has to report the non-existence of solutions to the problem. Although motion planning problems are interesting from a practical perspective, the problem is known to be computationally challenging for complete algorithms to generate solutions. In fact, a simple version of the problem without dynamics, referred to as the *Piano Mover's Problem*, was proven to be PSPACE-hard [97], which implies that complete algorithms are destined to suffer from computational complexity, as the system's degree of freedom increases.

A long-standing challenge for several decades has been designing computationally-efficient motion planning algorithms that can handle systems with high-dimensional configuration spaces. Particularly tailored for this setting, sampling-based algorithms such as PRMs [60, 18] have achieved great success in computationally challenging instances, as multi-query solutions that keep reusing pre-computed roadmaps for queries about different pairs of initial and goal configurations. Later, single-query algorithms such as RRTs [74, 65] have better addressed planning of systems with non-holonomic constraints, kinodynamic constraints, or uncertainties, especially in incremental and on-line settings. Most sampling-based methods, including PRMs and RRTs, gain computational tractability by relaxing the completeness requirement to *probabilistic completeness*, meaning that the probability of finding a solution, if one exists, converges to one as the number of samples tends to infinity. This probabilistically complete algorithm cannot report its failure or inability to solve infeasible problems. Another type of relaxation appears as a *resolution complete* algorithm, that finds an existing solution if the resolution of grids or samples is fine enough. A resolution complete algorithm cannot determine whether its failure originates from the infeasible problem or from the insufficient resolution. Resolution complete algorithms are more common with grids [106, 61], but a sequence of deterministic samples allows the adaptation with sampling-based approaches [25, 73].

Although PRMs and RRTs have managed to tackle previously challenging problems, none of the algorithmic variations in the past attain both the computational efficiency and the solution optimality [57]. As an optimal variant of RRTs, the RRT* [57] algorithm was proposed to ensure the *asymptotic optimality* of solutions while preserving the probabilistic completeness and the computational efficiency of RRTs. The RRT* enhances long-term behaviors of sampling-based algorithms with the asymptotic optimality, mathematically grounded on the Percolation theory [19] and Random Geometric Graphs [87].

The first part of this dissertation further improves the optimal motion planning algorithms, in terms of the convergence rates to the optimal solutions. For that matter, we formally define the optimal motion planning problem as follows. Let $X \subset \mathbb{R}^n$ and $U \subset \mathbb{R}^m$ be compact sets that represent state and input spaces. Let $X_{\text{obs}} \subset X$ and $X_{\text{free}} := X \setminus X_{\text{obs}}$ denote *obstacle region* and *obstacle-free region.* Let $U_{\text{feas}} \subset U$ and $X_{\text{feas}} \subset X$ be *feasible inputs* and *feasible states* that satisfy input and state constraints. Let $x_0 \in X$ be the initial state, and let $X_{\text{goal}} \subset X$ be a feasible goal region such that $X_{\text{goal}} \cap X_{\text{free}} \cap X_{\text{feas}} \neq \emptyset$. Then, a continuously differentiable function f with respect to both of its variables x and u such that

$$\dot{x}(t) = f(x(t), u(t)), \quad x(0) = x_0, \tag{1.1}$$

defines a time-invariant dynamical system. Let a *state trajectory* $\mathbf{x} := x(0 \leq t \leq T)$ and an *input trajectory* $\mathbf{u} := u(0 \leq t \leq T)$ be defined by a tuple $(\mathbf{x}, \mathbf{u}, T)$ that satisfies the function f over $t \in [0, T]$. More strictly, let the state and input trajectories consist of feasible states in the obstacle-free region and feasible inputs, i.e.,

$$\dot{x}(t) = f(x(t), u(t)), \quad x(t) \in X_{\text{free}} \cap X_{\text{feas}},\ u(t) \in U_{\text{feas}}. \tag{1.2}$$

Let $\mathcal{X}$ and $\mathcal{U}$ denote the set of $\mathbf{x}$ in $X_{\text{free}} \cap X_{\text{feas}}$ and the set of $\mathbf{u}$ in U_{feas} for all real values $T > 0$. The set of tuples $(\mathbf{x}, \mathbf{u}, T)$ is denoted $\mathcal{T}$ as *feasible trajectories.* Given Lipschitz continuous functions $g : (X, U) \to \mathbb{R}_{\geq 0}$ and $h : X \to \mathbb{R}_{\geq 0}$, and feasible trajectories $\mathbf{x} \in \mathcal{X}$ and $\mathbf{u} \in \mathcal{U}$, then a *cost functional*

$$J(\mathbf{x}, \mathbf{u}) = h(x(T)) + \int_0^T g(x(t), u(t))\, dt \tag{1.3}$$

associates each feasible trajectory $(\mathbf{x}, \mathbf{u}, T) \in \mathcal{T}$ with a cost, where T is free to choose.

Problem 1.1 (Optimal Motion Planning) *Given a state space X, an obstacle region X_{obs}, a feasible goal region X_{goal}, feasible states X_{feas}, feasible inputs U_{feas}, an initial state $x_0 \in X_{\text{feas}} \setminus X_{\text{obs}}$, a dynamical system described as in* (1.1)*, and a cost functional $J : \mathcal{T} \to \mathbb{R}_{\geq 0}$ described as in* (1.3)*, find a tuple $(\mathbf{x}, \mathbf{u}, T) \in \mathcal{T}$ that satisfies Equation* (1.1) *for all $t \in [0, T]$,*

- *starts from the initial state, i.e., $x(0) = x_0$,*
- *reaches the goal region, i.e., $x(T) \in X_{\text{goal}}$,*
- *avoids collision, i.e., $x(t) \notin X_{\text{obs}}$, $\forall t \in [0, T]$,*
- *respects constraints, i.e., $x(t) \in X_{\text{feas}}$, $u(t) \in U_{\text{feas}}$, $\forall t \in [0, T]$,*
- *and minimizes the cost functional J over the trajectory.*

In a similar setting, optimal feedback planning is defined as follows.

Problem 1.2 (Optimal Feedback Planning) *Given a state space X, an obstacle region X_{obs}, a feasible goal region X_{goal}, feasible states X_{feas}, a region of interest $Z \subset X_{\text{feas}} \setminus X_{\text{obs}}$, feasible inputs U_{feas}, a dynamical system described as in* (1.1)*, and a cost functional $J : \mathcal{T} \to \mathbb{R}_{\geq 0}$ described as in* (1.3)*, find a feedback policy $\pi : Z \to U_{\text{feas}}$ over the region of interest Z that*

- *outputs control inputs for the region of interest Z, i.e., $x(t) \in Z$, $u(t) \in U_{\text{feas}}$*
- *reaches the goal region, i.e., $x(T) \in X_{\text{goal}}$ where T is free to choose*
- *avoids collision, i.e., $x(t) \notin X_{\text{obs}}$, $\forall t \in [0, T]$,*

- *respects constraints, i.e.,* $x(t) \in X_{\text{feas}}$*,* $u(t) \in U_{\text{feas}}$*,* $\forall t \in [0, T]$*,*
- *and minimizes the cost functional* J *over the trajectory.*

Definition 1.3 (Optimal Cost-To-Go and Cost-To-Come) *In the optimal motion planning (Problem 1.1) or optimal feedback planning (Problem 1.2), let* $\bar{J}(z, z_f)$ *denote the optimal cost-to-go value (or the value function), i.e., the minimum of the cost functional* J *to reach the goal region starting from the state* z*. Similarly, let* $\bar{J}(z_i, z)$ *denote the optimal cost-to-come value, i.e., the minimum of the cost functional* J *to reach the state* z *starting from the initial state.*

Definition 1.4 (CostToCome and CostToGo Procedures) *In a forward tree with its root at an initial state, let the* `CostToCome` *procedure, or shortly the* `Cost` *in the forward tree, return the best cost-to-come value* $J(z_i, z)$ *by following tree edges. In a backward tree to goal states, let the* `CostToGo` *procedure, or shortly the* `Cost` *in the backward tree, return the best cost-to-go value* $J(z, z_f)$ *by following tree edges. We call a trajectory from* z_i *to* z *is optimal if* $\bar{J}(z_i, z)$ *is equal to* `Cost`(z) *in the forward tree. Similarly, we call a trajectory from* z *to* z_f *is optimal if* $\bar{J}(z, z_f)$ *is equal to* `Cost`(z) *in the backward tree.*

1.2 Planning with Dynamical Systems

In Problem 1.1 of optimal motion planning, the system is described as a differential equation (1.1), i.e., a general class of "dynamical systems". However, a substantial portion of previous efforts have considered optimal "path" planning problems, by focusing on trivial systems such as $\dot{x} = u$ [57, 6]. Motion planning problems with dynamical systems, optimal or sub-optimal, have remained challenging for realistic systems, particularly with non-holonomic constraints or kinodynamic constraints [30, 82, 70, 71, 74, 65]. Overall, a fairly limited amount of previous efforts have addressed optimal motion planning problems with "dynamical systems" [56, 55, 54]. In that regard, we emphasize our consideration of dynamical systems in optimal motion planning, by selecting the redundant title with "dynamical systems".

In this domain, most solution approaches involve relaxation of the optimality in several different ways. The only exception comes from the literature of optimal control theory [21], if the associated HJB equation is analytically solvable and the resultant trajectory happens to be collision-free. Once the set of equations becomes analytically intractable, resorting to numerical methods or optimization methods has been prevalent, e.g., as in numerical integration [93], Model Predictive Control (MPC) [22], constrained Linear Quadratic Regulator (LQR) [11], and other optimization-based approaches such as LQR-trees [108] and CHOMP [96]. Such approaches involve relaxation in numerically approximating the solution or in transcribing the original problem into optimization problems with a tractable number of variables. Unless the set of constraints induces a convex region, optimization-based approaches additionally involve relaxation into locally optimal solutions.

Relaxation into resolution complete algorithms allows optimal path planning algorithms in grids or lattices. Although Dijkstra [29] and A* [44] are originally designed to search paths in graphs, their application to discretized grids or lattices still generates solutions for optimal paths, as long as the underlying graph representation fully reflects the system's available actions or paths. Recent variants such as D*, Focused D*, and D*-Lite [106, 61] enhance their replanning performance by efficient and focused propagation of updated information. Apart from the well-known curse of dimensionality issue [10], we notice that the underlying graph representation of motions is often insufficient to express all possible evolutions of the system dynamics. Meshes and simplexes may be considered in place of grids [122, 121], but application examples tend to remain in the domain of paths, not of dynamical motions.

Markov Decision Processes (MDPs) [94], typically represented on grids, handle stochastic control problems. Due to the dependence on discretization, approximation, or interpolation of time, state space, input space, or value function, MDPs involve relaxation by fewer outcomes and less accurate transitions of states, than originally described by the dynamics. Moreover, the inclusion of discount factor $\gamma < 1$ in the formulation accelerates the convergence of the iterations, but results in an additional relaxation of uniformly weighted solutions into future discounted solutions, from the perspective of the cost functional J defined in (1.3).

Relaxation to probabilistically complete algorithms is available as well. Preliminary work on extending the RRT* algorithm to handle systems with differential constraints has appeared in [56], which discussed application examples such as double integrators and Dubins' vehicle dynamics [31]. Practical issues in replanning scenarios were addressed in [55], emphasizing the benefits of anytime algorithms. Additional issues with non-holonomic constraints were addressed in [54]. However, the current set of practical solutions remain fairly limited, thus we attempt to elaborate more on this probabilistically complete algorithms, mainly with non-trivial dynamical systems as the application examples.

1.3 Steering Methods

Steering problems, or Two-Point Boundary Value Problems (TPBVPs), take important roles as the crucial sub-routines in sampling-based motion planning algorithms [60, 18, 74, 65, 57].

Problem 1.5 (Steering Problem, or TPBVP) *Given a state space X, feasible states X_{feas}, feasible inputs U_{feas}, an initial state $x_0 \in X_{\text{feas}}$, a final state $x_f \in X_{\text{feas}}$, a dynamical system described as in* (1.1)*, find a tuple* $(\mathbf{x}, \mathbf{u}, T) \in \mathcal{T}$ *that satisfies* (1.1) *for all* $t \in [0, T]$,

- *starts from the initial state, i.e.,* $x(0) = x_0$,
- *reaches the final state, i.e.,* $x(T) = x_f$,
- *respects constraints, i.e.,* $x(t) \in X_{\text{feas}}$, $u(t) \in U_{\text{feas}}$, $\forall t \in [0, T]$.

Note that the problem definition does not require the steered trajectory to be collision-free, i.e., a state $x(t)$ is not compared with an obstacle region X_{obs}.

Problem 1.6 (Optimal Steering Problem, or Optimal TPBVP) *Given a cost functional* $J : \mathcal{T} \to \mathbb{R}_{\geq 0}$ *described as in* (1.3)*, find a tuple* $(\mathbf{x}, \mathbf{u}, T) \in \mathcal{T}$ *that satisfies all requirements in Problem 1.5,*

- *and minimizes the cost functional J over the trajectory.*

By definition, steering problems or TPBVPs effectively imply motion planning problems in an obstacle-free space. In the context of RRTs [74, 65, 76], the final state x_f needs not be reached exactly, thus steering problems do not necessarily mean TPBVPs. With other algorithms that involve graphs, roadmaps, or rewiring operations as in PRMs [60], the RRT* [57], and most of later algorithms, steering problems are assumed to connect the initial state x_0 and the final state x_f exactly, thus steering problems and TPBVPs become interchangeable.

In the absence of obstacle-induced constraints, solutions to certain classes of optimal TPBVPs can be discovered in the literature of optimal control theory [21]. In [56], the RRT* embeds the solutions for time-optimal TPBVPs of double integrators and Dubins' vehicle dynamics [31], for which the analytic optimal solutions are known. The infinite-horizon LQR is employed as a steering solution in [89], but the exactness of the final state x_f is exchanged with the ease of computing the optimum by the LQR solution. The follow-up work in [41] elaborates the finite-horizon LQR, but the exactness of x_f is still missing as briefly shown in [85]. A Gramian-based finite-horizon optimal TPBVP solution for linear systems in [117] minimizes the weighted cost of terminal time and control efforts, assuming the involved numerical integration does not incur numerical issues. As an extension to non-linear systems, a successive approximation approach in [43] obtains TPBVP solutions for the same cost to [117], whereas each TPBVP requires more computation than previous approaches.

Optimization-based solutions with high precisions can serve as good approximations to the optimal TPBVP solutions, with alleviated local minima issues for the overall trajectory. Note that a single optimization problem over the entire trajectory is distinguished from a concatenation of optimized trajectory segments. In this domain, our work in [51] numerically implements the methodology to obtain time-optimal trajectories. Later, the work in [118] employs optimization tools, i.e., sequential quadratic programming (SQP) [17], to solve optimal TPBVPs.

Our interests exist in expanding the library of efficient TPBVPs, suitable as a subroutine for the RRT* and other sampling-based algorithms. The aforementioned work in [51] targets general dynamical systems. The work in [49] exploits characteristics of dynamical systems, such as the differential flatness [35]. The work in [50] further utilizes the flatness and focuses on controllable linear systems.

1.4 Statement of Contributions

The main contributions of this dissertation are 1) practical and efficient adaptations of optimal sampling-based motion planning algorithms, and 2) inclusion of non-trivial dynamical systems as the subject of optimal motion planning. Our contributions on the planning framework are summarized as follows:

- Chap 2.1, 2.2: In basic versions [57] of the RRT*, RRG, and PRM*, algorithm descriptions are chosen to better address mathematical properties, than practical issues. Accordingly, a large amount of research efforts have been invested toward fast and efficient implementation, by the authors and by other researchers. Our proposed integration aims to obtain the upper-envelope performance of existing modifications, by carefully importing the key algorithmic components into the context of incremental algorithms, while preserving the ideal behaviors by the discussed algorithms.

- Chap 2.3: By simple and natural conversions of motion planning algorithms into the backward structure that is prevalent in the literature of dynamic programming, optimal control, and artificial intelligence, we enable feedback planning or efficient replanning that reuses most of the previous computation.

- Chap 2.4: We provide an efficient computation scheme for informed TPBVPs, with non-trivial dynamical systems and computationally demanding TPBVPs in consideration. Our modified scheme significantly reduces the amount of total computation needed for steering problems, thus more complicated dynamical systems can be handled within the same computational budget.

Sampling-based motion planning algorithms rely on specific steering methods, or TPBVP solvers. Our contributions in the domain of TPBVPs are as follows:

- Chap 3: For a general class of dynamical systems, we propose a numerical method for TPBVPs. Time-optimal off-road vehicle maneuvers are shown as the example.

- Chap 4: For pseudo-flat dynamical systems, we propose a semi-analytic solution for TPBVPs. Computation of time-optimal off-road vehicle maneuvers is expedited, and more complicated scenarios can be handled as the result.

- Chap 5: For controllable linear systems, we propose an efficient method for TPBVPs, based on the progressive and incremental computation of small-sized linear or quadratic programming problems.

1.5 Outline

This dissertation is organized as follows. Contributions on the motion planning framework are provided in Chapter 2, and the subsequent chapters describe specific TPBVP solutions. Chapter 3 discusses a numerical method for general dynamical systems. Chapter 4 proposes a semi-analytic solution for pseudo-flat dynamical systems. Chapter 5 further utilizes the differential flatness and formulates small-sized linear or quadratic programs for controllable linear systems, where the set of constraints grows progressively. This dissertation concludes with summary and remarks in Chapter 6.

Chapter 2

Enhanced RRT* Algorithms and Further Modifications for Dynamical Systems

Representative algorithms in the evolution of sampling-based motion planning, namely Probabilistic RoadMaps (PRMs) [60], Rapidly-Exploring Random Trees (RRTs) [74], and the RRT* algorithm [57], have mainly elaborated answers for two subproblems: 1) *neighbor selection* (how the subset of samples are selected for processing), and 2) *sample processing* (how the selected subset of samples are processed at iterations). How to generate low-discrepancy and low-dispersion samples beforehand, by random or deterministic sampling strategies, is a fairly decoupled topic despite its importance in sampling-based algorithms. Interested readers may refer [73] for details.

The first subproblem of neighbor selection with grids, lattices, or n samples in a d-dimensional space has mostly relied on proximity queries or range queries. Successful examples in the literature include adjacent (for grids or lattices) [29, 44, 106, 61], nearest [74], k-nearest (k may be a constant or proportional to $\log n$) [60, 57], and in-range (range may be a constant or proportional to $((\log n)/n)^{1/d}$) [60, 57] queries. Prior to the RRT*, the second subproblem of processing the selected samples used to be simpler, i.e., paired samples are assumed to be precisely connected [60, 57] or approached as closely as possible [74]. The RRT* involves one more crucial processing step of *rewiring* edges and vertices within a shrinking ball with its volume proportional to $(\log n)/n$, or equivalently with $\mathcal{O}(\log n)$ neighbors, to ensure the optimality of the solution in the limit. Rewiring the tree is equivalent to assigning a better parent vertex for the considered vertex, in a way the *cost-to-come* (the cost from the initial state to the current state) value to the vertex is minimized by rewiring to the candidate vertices within the considered volume proportional to $(\log n)/n$, or equivalently to $\mathcal{O}(\log n)$ neighboring vertices.

The procedure of precisely connecting the paired samples in a configuration space is called a *steering problem* in the literature, or equivalently a *two-point boundary value problem* (TPBVP) in the case of precise connection. The steering problem is trivially simple for trivial Euclidean systems $\dot{x} = u$, and the problem becomes fairly hard to solve for general dynamical systems. Algorithms that rely on inexact steering methods are out of the scope in this chapter, due the to lack of provable guarantees and analyses. However, inexact steering solutions based on forward simulations are often more convenient to obtain than exact TPBVP solutions. Interested readers may refer [76], or the preceding work of ours in [51].

Although the RRT* practically belongs to *anytime algorithms* [123, 124, 57, 55] that quickly obtain a first solution and continue to improve the solution toward the optimal one, the convergence to the optimum is reportedly slow. As mentioned, the execution further slows down when non-trivial steering solutions [51, 49] need to replace the straight paths, namely the steering solutions for trivial systems $\dot{x} = u$. Naturally, a significant amount of research efforts have proposed promising ideas for faster convergence than the RRT* [83, 6, 48, 90, 39, 101, 40, 52]. We note that the proposed modifications tend to remain as isolated algorithms with own benefits, where the algorithmic advantages are difficult to combine together.

Within our abstract algorithmic framework, we first interpret the key contributions of aforementioned efforts toward the faster convergence, and we lead to unified perspectives and modifications, i.e., the enhanced RRT* algorithm. In the context of feedback motion planning, we also propose a sampling-based solution on the tree data structure. In addition, a modification is proposed for the efficient inclusion of dynamical systems in sampling-based motion planning algorithms.

This chapter starts with literature reviews in Section 2.1. Higher-level discussions follow on grid-based algorithms and recent sampling-based algorithms that attempt to achieve faster convergence rates toward the optima than the RRT* algorithm. Contributions and implications of each approach are interpreted comparatively within a higher-level structure, showing that each approach can be interpreted as a variant with different sampling strategies and neighbor selection procedures. In Section 2.2, we propose the enhanced RRT* algorithm that natively combines key factors of such algorithmic variants within the RRT*'s simple procedures and $\mathcal{O}(\log n)$ complexity per iteration. Section 2.3 suggests a natural modification, named as Goal-Rooted Feedback Motion Trees (GR-FMTs), that efficiently generates feedback policies around motion plans toward goals, with computational complexity same to the RRT*'s. Section 2.4 substantially improves the loop of non-trivial steering attempts, i.e., when general dynamical systems are considered for motion planning problems. Section 2.5 compares our algorithmic variants in simulation experiments, and Section 2.6 concludes with remarks.

2.1 Toward Faster Convergence Rates than the RRT*

We begin by suggesting an abstract form of algorithms that can include grid-based and sampling-based algorithms as the subsets. Our literature review is comparative within the suggested structure, focusing on algorithms with proofs or claims for convergence to the optimal solution [29, 44, 106, 61, 57, 83, 6, 48, 90, 39, 101, 40, 52]. Because our coverage suffices to explain all major algorithmic components, we neither mention research efforts toward the relaxed optimality [3, 100] nor enumerate other later algorithms with provable optimality guarantees.

2.1.1 Abstract Form of Sampling-Based Algorithms

Algorithm 2.1 is an abstract form of sampling-based motion planning algorithms, allowing variations in the procedures `Sample`, `SelectNeighbors`, `ConnectToBestVertex`, and PropagateInfo. We force the entire set of vertices V_i and the entire set of edges E_i to constitute a tree G_i rooted at $V_{i=0}$(=Vertex(z_{root})), by maintaining the information about a single parent and children at each of the vertices V_i. In addition, following the ancestor vertices and edges in the forward tree, each vertex stores the cost-to-come value starting from the root V_0. Hence, a set of vertices $V_{i,\text{goal}}$ inside goal regions are equivalent to the set of trajectories from V_0 to goal regions and the corresponding trajectory costs. For convenience, G_i, V_i, E_i, $V_{i,\text{goal}}$, $V_{i,\text{revisit}}$, and $V_{i,\text{remain}}$ with subscripts i are treated as global variables at i-th iteration, thus any procedure can access the data for the current tree, vertices, edges, and goal-reaching trajectories. $V_{i,\text{revisit}}$ and $V_{i,\text{remain}}$ are reserved for later use. Variables with other explanatory subscripts remain local and temporary. For instance, z_{samp} and v_{samp} denote the sample state and the corresponding vertex at i-th iteration, and V_{near} denotes a subset of V_i near z_{samp}. Lower-case variables such as v_{new} and e_{new} contain only one element, i.e., an updated vertex from v_{samp} and an edge from v_{samp} to the best parent vertex found in V_{near}. No element may be returned as v_{new} or e_{new}, if all paths to near vertices in V_{near} are invalid due to obstacles or other violated constraints.

```
  Globally Accessible Data: $i$, $G_i$, $V_i$, $E_i$, $V_{i,\text{goal}}$, $V_{i,\text{revisit}}$, and $V_{i,\text{remain}}$
1 $i \leftarrow 0$; $V_i \leftarrow$ Vertex($z_{\text{root}}$); $E_i \leftarrow \emptyset$; $G_i \leftarrow (V_i, E_i)$; $V_{i,\text{goal}} \leftarrow \emptyset$;
2 while $i{+}{+} < N$ and Interrupted() = false do
3     $(z_{\text{samp}}, v_{\text{samp}}) \leftarrow$ Sample();
4     $V_{\text{near}} \leftarrow$ SelectNeighbors($z_{\text{samp}}$, backward_reachable);
5     $(v_{\text{new}}, e_{\text{new}}) \leftarrow$ ConnectToBestVertex($v_{\text{samp}}, V_{\text{near}}$);
6     $V_{\text{near}} \leftarrow$ SelectNeighbors($z_{\text{samp}}$, forward_reachable);
7     $G_{i+1} \leftarrow (V_{i+1}, E_{i+1}, V_{i+1,\text{goal}}) \leftarrow$ PropagateInfo($v_{\text{new}}, e_{\text{new}}, V_{\text{near}}$);
8 return Traj($V_{i,\text{goal}}$);
```

Algorithm 2.1: An Abstract Form of Sampling-Based Algorithms

Procedure names in Algorithm 2.1 and subsequent algorithms in this chapter are set to be straightforward. The procedures Vertex and Edge create a vertex and an edge respectively, provided sufficient information for the creation. The procedures State and Cost provide the state and the cost-to-come value of a vertex in inquiry. At each iteration unless interrupted, a sample z_{samp} and the corresponding vertex v_{samp} are created in `Sample`, and the set of neighbors V_{near} is selected in `SelectNeighbors`. Note that neighbors reachable to the randomly drawn sample within a time or cost horizon (or neighbors in the *backward reachable* set) generally differ from neighbors reachable from the sample within the same horizon (or neighbors in the *forward reachable* set). In `ConnectToBestVertex`, an edge e_{new} is created to connect v_{samp} to a vertex in V_{near} as a part of the tree G_i, in a way that the cost-to-come value to v_{samp} is minimized. In the `PropagateInfo` procedure, further processing such as the RRT*'s rewiring operation improves the underlying tree G_i into G_{i+1}.

As a simple example, it is trivial to shape RRTs [74] into a variant of Algorithm 2.1. By following the procedure names in [74], the `SelectNeighbors` procedure is replaced by the `NEAREST_NEIGHBOR`, and the `ConnectToBestVertex` procedure is roughly equivalent to the `EXTEND`. The `PropagateInfo` procedure is set to merely update global variables G_i, V_i, E_i, and $V_{i,\mathrm{goal}}$ for later iterations.

We reorder routines and reshape some of the algorithms in [29, 44, 106, 61, 57, 83, 6, 48, 90, 39, 101, 40, 52] into variants of Algorithm 2.1, mainly with different implementations of the procedures `Sample` and `SelectNeighbors`. The RRT* [57] serves as the baseline algorithm for the procedures `ConnectToBestVertex` and `PropagateInfo`.

2.1.2 The RRT* Algorithm [57]

A version of the RRT* algorithm is shown by assigning Algorithm 2.2 and 2.3 as the `ConnectToBestVertex` and `PropagateInfo` procedures. Contrary to the original representation of the RRT* in [57], this version does not construct the preliminary edge to the randomly drawn sample from the nearest vertex in the tree, between Line 3 and 4 in Algorithm 2.1. This version returns the identical tree to the original version if none of such preliminary edges are in collision. Otherwise, this version of the RRT* algorithm returns a tree with a larger number of vertices.

Algorithm 2.2 shows the `ConnectToBestVertex` procedure that creates the best edge e from a set of vertices V_{near} and creates a vertex v out of the inquired vertex v_{in}. As a generalized sub-routine, Algorithm 2.2 may be informed about J_{upper}, an upper bound on the cost of the potential edge e. Once the `TPBVP` procedure solves a TPBVP that steers the system from the state of a vertex v_{near} in V_{near} to the state of the inquired vertex v_{in}, then the generated state trajectory, the required control inputs, and the trajectory cost are stored in x_{near}, u_{near}, and J_{near}. A loop with the vertices in V_{near} keeps track of the best TPBVP solution, if safe from collisions with obstacles. The loop finally returns the best edge e that contains the best TPBVP solution and the created vertex v.

```
  Input  : $v_{\text{in}}$, $V_{\text{near}}$, and $J_{\text{upper}}$ ($= \infty$ if unspecified)
  Output: $v$ and $e$
1 $v \leftarrow \emptyset$; $e \leftarrow \emptyset$;
2 for all $v_{\text{near}} \in V_{\text{near}}$ do
3     $(x_{\text{near}}, u_{\text{near}}, J_{\text{near}}) \leftarrow$ TPBVP(State($v_{\text{near}}$), State($v_{\text{in}}$));
4     if Cost($v_{\text{near}}$) $+ J_{\text{near}} < J_{\text{upper}}$ and ObstacleFree($x_{\text{near}}$) then
5         $J_{\text{upper}} \leftarrow$ Cost($v_{\text{near}}$) $+ J_{\text{near}}$;
6         $v \leftarrow v_{\text{in}}$; $e \leftarrow$ Edge($v_{\text{near}}, v, (x_{\text{near}}, u_{\text{near}}, J_{\text{near}})$);
7 return $(v, e)$;
```

Algorithm 2.2: The `ConnectToBestVertex` Procedure for the RRT*

```
  Input  : $v$, $e$, and $V_{\text{near}}$
  Output: $V_{i+1}$, $E_{i+1}$, and $V_{i+1,\text{goal}}$
1 $V_{i+1} \leftarrow V_i \cup \{v\}$; $E_{i+1} \leftarrow E_i \cup \{e\}$; $V_{i+1,\text{goal}} \leftarrow V_{i,\text{goal}} \cup$ VertexInGoals($v$);
2 for all $v_{\text{near}} \in V_{\text{near}}$ do
3     $(v_{\text{rewire}}, e_{\text{rewire}}) \leftarrow$ ConnectToBestVertex($v_{\text{near}}$, $\{v\}$, Cost($v_{\text{near}}$));
4     if $v_{\text{rewire}} \neq \emptyset$ and $e_{\text{rewire}} \neq \emptyset$ then
5         $E_{i+1} \overset{-}{\leftarrow}$ Edge(Parent($v_{\text{near}}$), $v_{\text{near}}$); $E_{i+1} \overset{+}{\leftarrow} e_{\text{rewire}}$;
6         $V_{i+1} \leftarrow$ RewireTreeVertices($v_{\text{near}}, v_{\text{rewire}}$);
7 return $(V_{i+1}, E_{i+1}, V_{i+1,\text{goal}})$;
```

Algorithm 2.3: The `PropagateInfo` Procedure for the RRT*

Algorithm 2.3 is the representation for the RRT*'s rewiring operation within the set of vertices in V_{near}. The `VertexInGoals` procedure returns the inquired vertex v

itself if the vertex is inside goal regions. Algorithm 2.3 generalizes the rewiring operation using the `ConnectToBestVertex` procedure (Algorithm 2.2) as a sub-routine. Note that $\{v\}$ contains only one element, thus `ConnectToBestVertex` does not contain a loop in computation. The existing edge in the tree from $\texttt{Parent}(v_{\text{near}})$ to v_{near} is removed and re-routed via e_{rewire} if the cost-to-come value to v_{near} can be improved by switching the parent vertex of v_{near} into v. The `RewireTreeVertices` procedure updates the existing vertex v_{near} and its descendants with the information of the new parent vertex v and the improved cost-to-come value to each vertex.

The RRT* algorithm (Algorithm 2.1, 2.2, and 2.3 with the `SelectNeighbors` procedure that returns $\mathcal{O}(\log n)$ near neighbors) is asymptotically optimal, i.e.,

$$\mathbb{P}\left(\left\{\limsup_{i\to\infty} \texttt{Cost}^{\text{RRT}^*}(V_{i,\,\text{goal}}) = c^*\right\}\right) = 1,$$

where c^* is the optimal cost to reach a state in goal regions from the initial state.

2.1.3 Canonical Modifications for the RRT*

Algorithm 2.2 and Algorithm 2.3 represent the most basic representation for the RRT*, focusing on the clear explanation of main ideas. In practice, there exist several modifications that are considered canonical to speed up the RRT* [55, 2, 51] since the inception of the RRT* algorithm. We elaborate a few widely-used modifications for the speed-up, that do not sacrifice the solution quality at all. Note that we may not explicitly write these modifications in the description of algorithms, to concentrate on more important procedures without potential confusions.

Branch and Bound

The branch and bound algorithm [68] is a well-known technique in combinatorial optimization. Robotics researches often implement the idea in conjunction with graph search algorithms [99, 67]. The idea of avoiding any computation that cannot improve the existing solution leads to efficient algorithms, and many successful algorithms such as A* [44] and k-d tree [12] are grounded on the same philosophy. The earliest applications of the branch and bound technique on the RRT* can be found in [55, 2, 51], and later work in [6, 90, 39, 101, 40] shape the same idea in different names and contexts, e.g., pruning, rejection, bounded propagation, and informed sampling.

An *admissible heuristic* is an estimate of a function that maps each state z to a non-negative real number, where the estimate is less than or equal to the actual value of the function. We consider admissible heuristics for both the cost-to-come value and the *cost-to-go* value, i.e., the cost of the optimal trajectory that reaches goal regions starting from z. A better approximation of the actual value leads to more effective pruning of tree branches and rejection of samples, but most often, exact functions are computationally expensive to get. In practice, even loose approximations enable significant reduction of the computation by pruning and rejection. For instance, the Euclidean distance from z to goal regions can serve as the lower bound on the actual trajectory length for a majority of dynamical systems.

We pretend the cost-to-go function is available for the simplicity of representation. Let procedures $\texttt{CostToGo}^*$ and $\texttt{Cost}^*$ for a state z or an existing vertex $v(=\texttt{Vertex}(z))$ implement the admissible heuristic of the cost-to-go function and the cost-to-come function. Let procedures `CostToGo` and `Cost` for an existing vertex v compute the cost-to-go from v and the cost-to-come to v, following edges on the tree. Let v_{goal} be the lowest-cost vertex in goal regions, then the branch and bound technique is the application of the following inequalities:

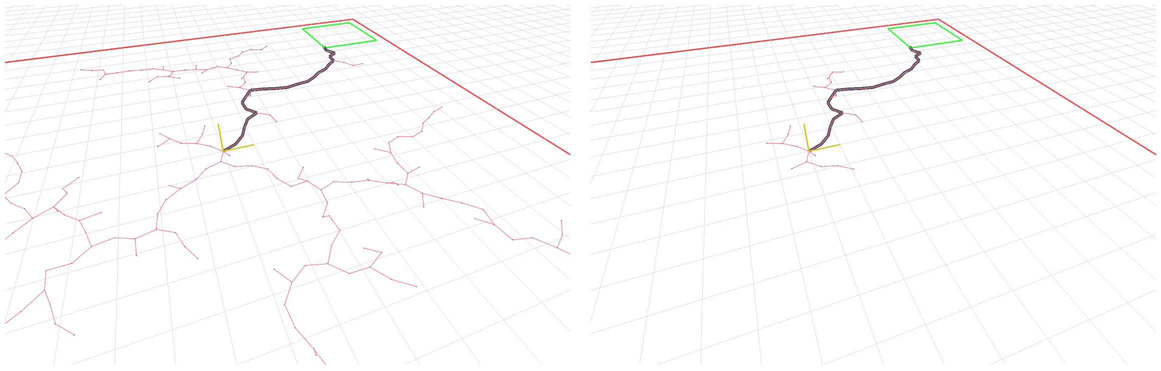
$$\texttt{Cost}(v) + \texttt{CostToGo}^*(v) > \texttt{Cost}(v_{\text{goal}}), \tag{2.1}$$

$$\texttt{Cost}^*(z) + \texttt{CostToGo}^*(z) > \texttt{Cost}(v_{\text{goal}}). \tag{2.2}$$

A state z or an existing vertex $v(=\texttt{Vertex}(z))$ that satisfies (2.2) can be safely removed or rejected, without losing a potentially better solution. If the state z is a newly drawn sample, an algorithm design may choose to project the sample to promising regions such that

$$\texttt{Cost}^*(z) + \texttt{CostToGo}^*(z) < \texttt{Cost}(v_{\text{goal}}), \tag{2.3}$$

instead of rejecting the sample.



(a) Before Pruning the Tree (b) After Pruning the Tree

Figure 2-1: Consequences of the Branch and Bound Technique

For an existing vertex v, any computation with (2.1) can be avoided permanently or temporarily, depending on the closeness between the values $\texttt{Cost}(v)$ and $\texttt{Cost}^*(v)$. More specifically, the inequality (2.1) solely implies that the vertex v in the current tree cannot contribute to a better solution. Provided that the inequality (2.2) does not hold, however, the same vertex v in the future tree may or may not contribute to a better solution than the current one. Therefore, whether to permanently remove or temporarily ignore such an existing vertex in the tree is subject to the preference.

The main benefit of the branch and bound technique is the reduced computation for proximity queries, e.g., $\mathcal{O}(\log n)$ for the k-d tree, where n is the number of vertices. Figure 2-1 illustrates such benefits and consequences of the branch and bound technique at a glance.

Remark 2.1 (Informed sampling) *The Informed RRT* [39] utilizes the branch and bound technique for direct sampling within a shrinking ellipsoidal region, computed by the inequality* (2.1) *using the admissible heuristic and the best found cost to goals. In [39], use of the cost functional based on the Euclidean metric between states leads to the computation of the ellipsoidal region. Success rates by informed sampling in a less cluttered environment are certainly higher than rejection sampling, whereas the benefit vanishes as the ellipsoidal region becomes bigger than the environment of interest in an extremely cluttered environment, e.g., a maze.*

For cost functionals in general forms, i.e., with dynamical systems, approximation *of the ellipsoidal region for informed sampling becomes challenging, due to the large difference between the Euclidean metrics and the pseudo-metrics. Instead, we may choose to reject samples based on the inequality* (2.2) *and project the samples toward promising regions.*

Delayed or Lazy Computation

Reordering, delaying, and avoiding more expensive computation until the last moment have been another source of efficient algorithms in the literature. Sampling-based motion planning algorithms have successfully adapted the idea, mostly by minimizing the frequency of collision checks for trajectories [18, 67, 88, 48, 45]. In a loop, candidates need to be sorted to enable delayed or lazy computation for expensive operations.

```
   Input  : $v_{\text{in}}$, $V_{\text{near}}$, and $J_{\text{upper}}$ ($= \infty$ if unspecified)
   Output: $v$ and $e$
1  $v \leftarrow \emptyset$; $e \leftarrow \emptyset$;
2  for all $v_{\text{near}} \in V_{\text{near}}$ do
3      $(x_{\text{near}}, u_{\text{near}}, J_{\text{near}}) \leftarrow \texttt{TPBVP}(\texttt{State}(v_{\text{near}}), \texttt{State}(v_{\text{in}}))$;
4      if $\texttt{Cost}(v_{\text{near}}) + J_{\text{near}} < J_{\text{upper}}$ then
5          $(E_{\text{sort}}, J_{\text{sort}}) \overset{+}{\leftarrow} (\texttt{Edge}(v_{\text{near}}, v_{\text{in}}, (x_{\text{near}}, u_{\text{near}}, J_{\text{near}})), \texttt{Cost}(v_{\text{near}}) + J_{\text{near}})$;
6  for all $(e_{\text{sort}}, J) \in \texttt{Sort}(E_{\text{sort}}, J_{\text{sort}})$ do
7      if $\texttt{ObstacleFree}(\texttt{Trajectory}(e_{\text{sort}}))$ then
8          $v \leftarrow v_{\text{in}}$; $e \leftarrow e_{\text{sort}}$;
9  return $(v, e)$;
```

Algorithm 2.4: A Lazy Version of the `ConnectToBestVertex` Procedure

Similar to [67, 88], a version of lazy computation for the `ConnectToBestVertex` procedure is shown in Algorithm 2.4. Assuming TPBVPs are simpler to compute, all TPBVPs are solved and sorted by the cost-to-come value to the vertex v_{in}, to minimize the number of calls for the `ObstacleFree` procedure. Note that TPBVPs for dynamical systems may become more expensive to compute than collision checks, then lazy computation for collision checking unnecessarily slows down the algorithm by prioritizing more expensive computation over simpler operations. Careful observation is required before modifying the algorithm with delayed or lazy computation.

Biased Samples

An occasional sampling bias toward goal regions, before finding the first solution that reaches the goal regions, has been a widely-accepted modification for RRTs [74], thus its inclusion in the RRT* has been inherent [2, 50]. Moreover, a local bias toward the found trajectory reportedly accelerates the convergence to the optimum [2, 83, 50]. If such types of biases become excessive, local minima issues, analogous to optimization-centric approaches, may be induced as the trade-off. This modification is simple to implement inside the `Sample` procedure of Algorithm 2.1, possibly incorporated with preset or adaptive weightings for different types of biases.

Segmentation or Truncation of Lengthy Trajectories

In earlier phases of iterations, samples and vertices are distant from the nearest or near neighbors. Thus, it is advisable to either divide a long trajectory into segments or limit the maximum length of trajectories, especially if near neighbors are searched by range, not in the ascending order of proximities. Although these minor modifications are rarely mentioned in the literature [50], the level of exploratory behaviors, especially in earlier phases, is affected by such modifications. A strategy that divides a long trajectory into overly small segments quickly increases the number of vertices in the tree, thus suffers from computational issues. In addition, many locally-biased vertices along the found trajectories may attract the solution to local minima, similar to optimization methods. Inspired by the RRT*'s rewiring ball radius, we advocate the division of a lengthy trajectory based on the value $((\log n)/n)^{1/d}$ that decreases as n increases. This segmentation is a counterpart to the strategy in the RRT-connect algorithm [65] that tries the previous control input again, upon its success, for the next edge expansion. Remind that RRTs do not assume exact TPBVP solutions, whereas the RRT* assumes and utilizes the exact TPBVP solutions.

Conditional Activation of Algorithmic Components

As shown with the branch and bound technique, quicker finding of a feasible solution is beneficial, in order to focus subsequent computation into promising areas. One simple but generic way to enforce exploratory behaviors is suppressing the loop of rewiring operations in Algorithm 2.3. Another level of exploratory behaviors is enabled by finding the nearest neighbor instead of near neighbors in the `SelectNeighbors` procedure. In other words, we may run RRTs or the RRT* with no rewiring operationn, until a first feasible solution is found. The conditional activation of algorithmic components in the RRT* is supported by benchmark results [40] where RRTs are consistently quicker in generating the first solution than other optimal variants of the RRT*.

Efficient Collision Checking and Free-Space Sampling

Readers may get benefits by referring the efforts for efficient computation of collision-related operations in [16]. On the contrary, this dissertation largely focuses on the computational modifications related to dynamical systems.

2.1.4 Graph-Based Propagation of Updated Information

Aforementioned modifications for the RRT* in Section 2.1.3 are largely inspired by or imported from tree-based algorithms. By construction, the RRT* is an extracted tree from the underlying graph, i.e., the Rapidly-Exploring Random Graph (RRG) [57]. Meanwhile, a notable perspective in [6] points out that the RRT* is not the best tree out of the underlying RRG. More specifically, the RRT* maintains the tree-based data structure, not the graph-based structure, thus any updated information by rewiring operations is exclusively propagated to descendants of the rewired vertex by the `RewireTreeVertices` procedure in Algorithm 2.3. As the outcome, non-descendant neighbors to the rewired vertex and their descendants cannot receive the updated information, and missed opportunities for potential re-routing by such neighbors differentiate the RRT* and the best possible tree extractable from the RRG.

Remark 2.2 (Asymptotic optimality of the RRG [57] and the RRT$^{\#}$ [6]) *The*

RRG algorithm and the RRT$^{\#}$ algorithm (in our representation, Algorithm 2.1, 2.2, 2.3, 2.5, and 2.6 with the `SelectNeighbors` *procedure that returns $\mathcal{O}(\log n)$ near neighbors) are asymptotically optimal, i.e.,*

$$\mathbb{P}\left(\left\{\limsup_{i\to\infty}\,\texttt{Cost}^{\mathrm{RRG}}(V_{i,\,\mathrm{goal}}) = c^*\right\}\right) = \mathbb{P}\left(\left\{\limsup_{i\to\infty}\,\texttt{Cost}^{\mathrm{RRT}^{\#}}(V_{i,\,\mathrm{goal}}) = c^*\right\}\right) = 1,$$

where c^* is the optimal cost to reach a state in goal regions from the initial state.

Remark 2.3 (Maximal propagation of updated information over the graph) *Given the same sequence of samples during i iterations,*

$$c^* \le \texttt{Cost}^{\mathrm{RRG}}(V_{i,\,\mathrm{goal}}) = \texttt{Cost}^{\mathrm{RRT}^{\#}}(V_{i,\,\mathrm{goal}}) \le \texttt{Cost}^{\mathrm{RRT}^*}(V_{i,\,\mathrm{goal}}) \le \texttt{Cost}^{\mathrm{RRT}}(V_{i,\,\mathrm{goal}}).$$

Note that the inequality holds at i-th iteration [6], not necessarily after spending the same amount of computation.

Compared to the RRT*, the RRT$^{\#}$ algorithm requires extra computation inside the `RewireTreeVertices` *procedure, while propagating the updated information to neighbors in the underlying graph by similar recursions to the Lifelong Planning A* (LPA*) [62]. The RRT$^{\#}$ may be understood as an efficient algorithm that maintains the minimum spanning tree (MST) [42] from the RRG, rooted at the initial state.*

Given a new sample (the red vertex located at the center of a ball), Figure 2-2 compares the tree-based propagation by the RRT* and the graph-based propagation by the RRT$^{\#}$, with respect to the updated information triggered by a successful rewiring operation for the green vertex. Dotted lines do not belong to edges in the tree of either the RRT* or the RRT$^{\#}$, but comprise parts of edges for the underlying RRG. As designed, the RRT* iteration connects the new sample to the existing tree and rewires the tree, particularly the green vertex in the illustrated example. In addition to the RRT* iteration, the RRT$^{\#}$ in Figure 2-2c recursively but efficiently propagates the updated information to non-descendant neighbors as well, e.g., the bottom-right vertex in magenta, thus the magenta vertex switches its parent vertex to lower its cost-to-come value.

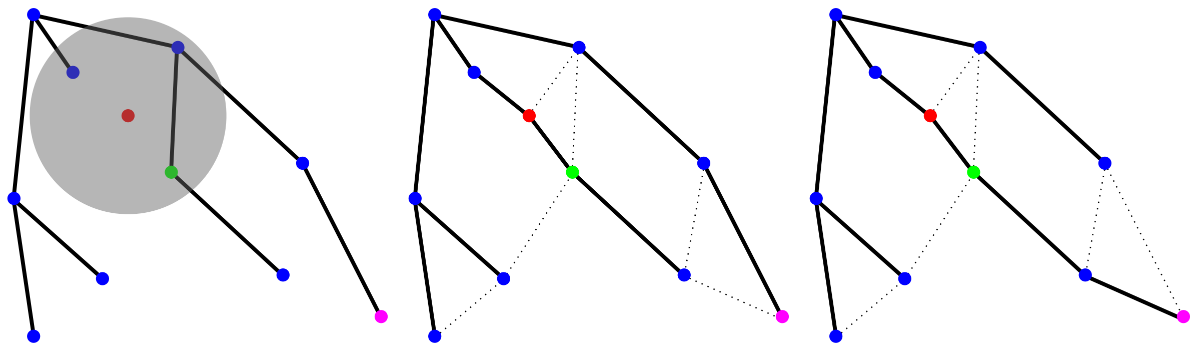

(a) New Sample (Red Vertex) (b) After the RRT* Iteration (c) After the RRT$^\#$ Iteration

Figure 2-2: Illustration of the RRT* and the RRT$^\#$

There exist trade-offs between graph-based operations and tree-based operations. Starting from a sample, the RRT* finishes the iteration earlier than the RRT$^\#$ and draws a new sample for the next iteration, at the same moment the RRT$^\#$ still propagates updated information more thoroughly over the graph. On the other hand, exploiting the thorough knowledge of vertices and edges in the graph is often more beneficial than adding new samples to the tree. For instance, the operation by the RRT$^\#$ in [6] tends to report faster convergence rates to the optimum than the RRT*. While the LPA*-like recursions are partly achieved by the branch and bound technique, the idea of graph-based propagation by the RRT$^\#$ often remains more rewarding than the RRT*, unless the sampling strategy is able to adapt the next sample in a way the same or larger improvement of the tree can be expected.

We attempt to build a bridge from the RRT$^\#$ (graph-based) to the RRT* (tree-based) within the abstraction Algorithm 2.1, by re-designing the `Sample` procedure. Let the `RewireTreeVertices` procedure (Algorithm 2.3) fill in the globally accessible data $V_{i,\text{revisit}}$ with non-descendant neighbors of the rewired vertex. Algorithm 2.5 shows the overridden definition of the `RewireTreeVertices` procedure for the RRT$^\#$, where the `UpdateDescendants` procedure in Line 1 is identical to the `RewireTreeVertices` procedure for the RRT*. As proposed, the remainder of Algorithm 2.5 adds non-descendant neighbors of the rewired vertex $v_{\text{old}}(=v_{\text{rewire}})$ into the set of vertices $V_{i,\text{revisit}}$ if the cost-to-come value can be improved by switching its parent vertex (Line 5). Based on the expected total cost to goal regions via each vertex, the set $V_{i,\,\mathrm{revisit}}$ is sorted for later use by the `Sample` procedure (Algorithm 2.6). For simpler representation, we first update the rewired branch of the tree in Line 1, and later check non-descendant neighbors for potential improvement. To prevent repeated updates over large branches of the tree, concerned users may propagate the updated information up to a single depth and frequently sort the list of candidates to be revisited.

```
  Input  : $v_{\mathrm{old}}$ and $v_{\mathrm{rewire}}$
  Output: $V_{i+1}$ and $V_{i+1,\,\mathrm{revisit}}$
1 $V_{i+1} \leftarrow \texttt{UpdateDescendants}(v_{\mathrm{old}}, v_{\mathrm{rewire}})$; $V_{i+1,\,\mathrm{revisit}} \leftarrow V_{i,\,\mathrm{revisit}}$;
2 $V_{\mathrm{near}} \leftarrow \texttt{SelectNeighbors}(\texttt{State}(v_{\mathrm{old}}), \mathrm{forward\_reachable})$;
3 for all $v_{\mathrm{near}} \in V_{\mathrm{near}} \setminus \texttt{Children}(v_{\mathrm{old}})$ do
4     $(x_{\mathrm{near}}, u_{\mathrm{near}}, J_{\mathrm{near}}) \leftarrow \texttt{TPBVP}(\texttt{State}(v_{\mathrm{old}}), \texttt{State}(v_{\mathrm{near}}))$;
5     if $\texttt{Cost}(v_{\mathrm{near}}) > \texttt{Cost}(v_{\mathrm{rewire}}) + J_{\mathrm{near}}$ then
6         $(V_{i+1,\,\mathrm{revisit}}, J_{\mathrm{revisit}}) \overset{+}{\leftarrow} (v_{\mathrm{near}}, \texttt{Cost}(v_{\mathrm{rewire}}) + J_{\mathrm{near}} + \texttt{CostToGo}^*(v_{\mathrm{near}}))$;
7 $\texttt{Sort}(V_{i+1,\,\mathrm{revisit}}, J_{\mathrm{revisit}})$;
8 return $(V_{i+1}, V_{i+1,\,\mathrm{revisit}})$;
```

Algorithm 2.5: The `RewireTreeVertices` Procedure for the RRT$^{\#}$

```
  Output: $z_{\mathrm{samp}}$ and $v_{\mathrm{samp}}$
1 if $V_{i,\,\mathrm{revisit}} = \emptyset$ then
2     $z_{\mathrm{samp}} \leftarrow \texttt{SampleRandom}()$; $v_{\mathrm{samp}} \leftarrow \texttt{Vertex}(z_{\mathrm{samp}})$;
3 else
4     $v_{\mathrm{samp}} \leftarrow \texttt{PopTheBest}(V_{i,\,\mathrm{revisit}})$; $z_{\mathrm{samp}} \leftarrow \texttt{State}(v_{\mathrm{samp}})$;
5 return $(z_{\mathrm{samp}}, v_{\mathrm{samp}})$;
```

Algorithm 2.6: The `Sample` Procedure for the RRT$^{\#}$

The `Sample` procedure in Algorithm 2.6 interprets and implements the graph-based propagation of the RRT$^{\#}$ as the revisit to the set $V_{i,\,\mathrm{revisit}}$ updated in Algorithm 2.5. The `PopTheBest` procedure returns a vertex v_{samp} with the lowest expected cost in the set $V_{i,\,\mathrm{revisit}}$ and removes the vertex from the set $V_{i,\,\mathrm{revisit}}$. For clear representation of main ideas, duplicate calls for SelectNeighbors and TPBVP remain in descriptions of Algorithm 2.1, 2.2, and 2.5. Note that such inefficiencies are simple to bypass or remove in implementation. Our representation of the RRT$^{\#}$ is an intermediate step toward our enhanced RRT*.

Proposition 2.4 (Equivalence of the RRT$^{\#}$ [6] and our representation) *Let our representation of the RRT$^{\#}$ be the composition of Algorithm 2.1, 2.2, 2.3, 2.5, and 2.6 with the* SelectNeighbors *procedure that returns $\mathcal{O}(\log n)$ near neighbors. Then, given the same sequence of random samples, the RRT$^{\#}$ [6] and our representation of the RRT$^{\#}$ algorithm find the same trajectory from the initial state to goals.*

Proof (Sketch) The formulation in [6] maintains locally minimum cost-to-come estimates (lmc values). A vertex is defined *stationary* if the lmc value matches with the cost-to-come value on the underlying RRG. A queue is maintained for non-stationary vertices, and LPA*-like recursions make each vertex in the queue stationary.

Our Cost procedure returns the cost-to-come value by following the existing tree. After a successful rewiring, discrepancies between our Cost procedure and the cost-to-come value on the underlying RRG are invoked by non-descendant neighbors of the rewired vertex. Our maintenance and revisit to $V_{i,\,\text{revisit}}$ make such vertices stationary and eliminate discrepancies between the tree and the graph. $\square$

To our best knowledge, both the graph-based propagation of updated information by the RRT$^{\#}$ [6] and our interpretation of the RRT$^{\#}$ as a revisiting procedure to vertices in the queue $V_{i,\,\text{revisit}}$ inside the Sample procedure are new perspectives in the literature. The RRT$^{\#}$ propagates the updated information more thoroughly than the RRT*, and our interpretation imports the idea into the context of tree-based algorithms. Later than the RRT$^{\#}$, Sampling-Based A* (SBA*) algorithms [90] adapt the connection to $\mathcal{O}(\log n)$ neighbors as a generalization of A* to sampling-based algorithms. By propagating updates on graphs, SBA* algorithms implicitly achieve similar benefits to the RRT$^{\#}$ starting from the A*'s point of view. Variations of SBA* mainly focus on generating samples that balance between exploration and exploitation, informed by previous iterations and current distributions.

Readers should note that the RRT$^{\#}$ obtains same solutions to the RRG and better solutions than the RRT*, with an expense of maintaining the graph, instead of the tree. The number of edges for the RRT$^{\#}$ increases as $\mathcal{O}(n \log n)$, not as $\mathcal{O}(n)$. In our representation that avoids storing the entire edges in the graph, the expense appears as extra calls for `SelectNeighbors` and `TPBVP` procedures on demand. We attempt to absorb the perspective into our re-designed sampling procedure.

2.1.5 Batch Processing

The choice in computation between batch processing and incremental processing may remain as a preference, not a general answer, especially given different circumstances or under uncertainties on computational budgets. Benchmarks may not directly compare these two different processing styles, reflecting that batch processing generates a data point after an execution whereas incremental processing provides a trajectory of data points during the execution. However, if the computational budget and the required computation for the batch are assumed to be precisely known, benchmark results in the Fast Marching Tree algorithm (FMT*) [48] suggest that batch processing gains computational benefits in certain circumstances. Inspired by the FMT*, Batch Informed Trees (BIT*) [40] progressively generates new batch of samples, trying to tune the algorithm between batch and incremental algorithms. We attempt to incorporate beneficial behaviors of batch algorithms into our incremental algorithm.

Remark 2.5 (Asymptotic optimality of batch algorithms) *The PRM* [57], the single-batch BIT* [40], and the FMT* [48] algorithms are asymptotically optimal, i.e.,*

$$\mathbb{P}\left(\left\{\limsup_{n\to\infty} \texttt{Cost}^{\mathrm{PRM}^*}(V_{n,\,\mathrm{goal}}) = c^*\right\}\right) = \mathbb{P}\left(\left\{\limsup_{n\to\infty} \texttt{Cost}^{1-\mathrm{BIT}^*}(V_{n,\,\mathrm{goal}}) = c^*\right\}\right) = 1,$$
$$\mathbb{P}\left(\left\{\lim_{n\to\infty} \texttt{Cost}^{\mathrm{FMT}^*}(V_{n,\,\mathrm{goal}}) > (1+\epsilon)c^*\right\}\right) = 0 \text{ for all } \epsilon > 0,$$

where $V_{n,\,\mathrm{goal}}$ is the set of vertices in goal regions after processing a batch of n samples, and c^* is the optimal cost to reach a state in goal regions from the initial state.

According to the statement in [48], the asymptotic optimality of the FMT* is math-

ematically weaker than the PRM and the single-batch BIT*, due to its lazy collision checks toward the best open neighbor vertex only. However, the relative weakness in the convergence may not matter in practice as claimed in [48], provided a limited number of samples.*

The single-batch BIT with a batch of infinitely many samples, different from the claim in [40] but clarified to a stronger claim, obtains the same solution with the PRM*, not the FMT*, owing to its more exhaustive connections than the FMT* to both open and closed neighbor vertices.*

Remark 2.6 (Extracted tree from roadmap) *With the same batch of n samples,*

$$c^* \leq \texttt{Cost}^{\mathrm{PRM}^*}(V_{n,\,\mathrm{goal}}) = \texttt{Cost}^{1-\mathrm{BIT}^*}(V_{n,\,\mathrm{goal}}) \leq \texttt{Cost}^{\mathrm{FMT}^*}(V_{n,\,\mathrm{goal}}),$$
$$c^* \leq \texttt{Cost}^{\mathrm{PRM}^*}(V_{n,\,\mathrm{goal}}) = \texttt{Cost}^{1-\mathrm{BIT}^*}(V_{n,\,\mathrm{goal}}) \leq \texttt{Cost}^{\mathrm{k-PRM}}(V_{n,\,\mathrm{goal}}),$$

where k for the k-nearest PRM is smaller than the number $\mathcal{O}(\log n)$ used by other optimal variants. Note that the inequalities hold after processing a batch of n samples, not necessarily after spending the same amount of computation.

The single-batch BIT extracts the minimum spanning tree from the underlying roadmap PRM*. The FMT* may be understood as an algorithm that lazily extracts its tree from the PRM*. Its extracted tree rooted at the initial state is not the minimum spanning tree out of the roadmap PRM*.*

Proposition 2.7 (Sub-optimality of grid-based approaches) *The application of Dijkstra [29], A* [44], D* [106], D*-Lite [61], and other optimal search algorithms on grids, with virtual connections to adjacent cells only, generates sub-optimal paths if the number of adjacent cells k is smaller than the number of available control actions and the number $\mathcal{O}(\log n)$ determined in [57].*

Proof (Sketch) The proof is based on the sub-optimality of k-nearest PRM [57], and the batch of samples is generated at each center of cells in grids. □

Proposition 2.8 (Asymptotic optimality of grid-based approaches) *The application of Dijkstra [29], A* [44], D* [106], D*-Lite [61], and other optimal search* *algorithms on grids, with virtual connections to k nearby cells, generates asymptotically optimal paths if k is determined by the number $\mathcal{O}(\log n)$ as in [57].*

Proof (Sketch) The proof is based on the asymptotic optimality of the PRM* [57], and the batch of samples is generated at each center of cells in grids. $\square$

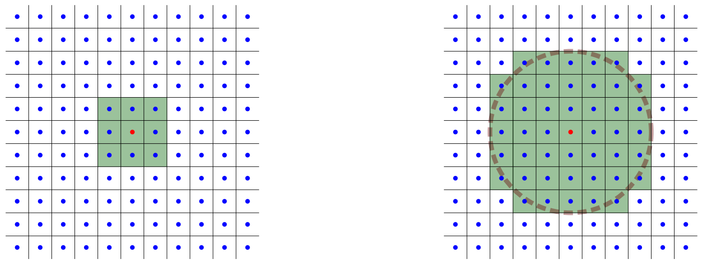

(a) Adjacent ($k = 8$) (b) Near ($k \propto \log n$)

Figure 2-3: Illustration of Adjacent Cells and Near Cells

Algorithms such as Dijkstra [29], A* [44], D* [106], and D*-Lite [61] are optimal in searching for the best path on the inquired graph. In Proposition 2.7, the source of the sub-optimality is the representation of grids with connections to adjacent cells only. The Theta* algorithm [28] is one of the research efforts to resolve the issue by shortcuts or smoothing, thus obtaining more straight paths. Results in [57] and Proposition 2.8 suggest more fundamental remedies, adapted by SBA* [90] (mentioned in Section 2.1.4) as a generalization of A* to sampling-based algorithms.

Queries for adjacent cells are efficiently computed if all cells are allocated in the data structure that scales exponentially in dimension and the degree of discretization. On the contrary, the data structure for proximity queries is incremental as the number of vertices increases, while the complexity of near-neighbor queries scales as $\mathcal{O}(\log n)$. Although we advocate the incremental use of both computation and storage for practical issues, it is noticeable that grid-based algorithms can attain the asymptotic optimality as well, but by considering the increased number of nearby cells $k(\propto \log n)$ for the finer discretization of grids. Figure 2-3 illustrates the difference.

Figure 2-4 illustrates the behaviors by the RRT* and the FMT* for the bug trap example, assuming the same set of samples for both, but with an unfortunate sequence of samples for the RRT*. Given initial and goal samples (red) along with a batch of samples (blue), the FMT* finds its way out from the trap. On the contrary, the RRT* may receive rare samples along the narrow corridor at early iterations, thus fails to grow the tree outward. Consequently, samples outside the trap remain wasted due to the attempts to connect with vertices inside the trap as dotted lines in Figure 2-4a.

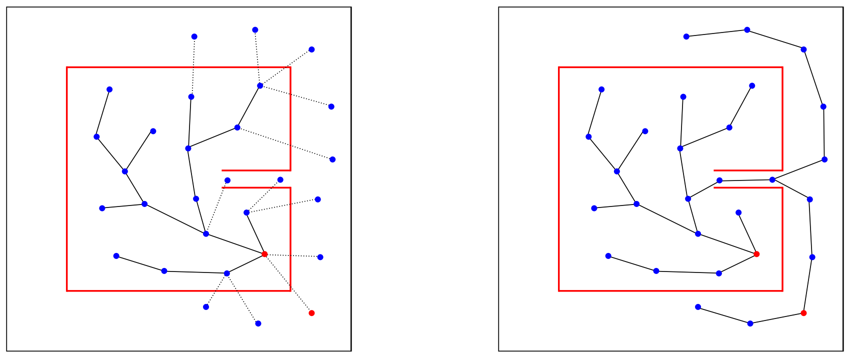

(a) The RRT* (Bad Case)   (b) The FMT*

Figure 2-4: The Bug Trap Example with the RRT* and the FMT*

The bug trap example suggests the importance of rare samples, particularly in the presence of narrow passages. Batch algorithms are designed for the maximal use of drawn samples, thus narrow passages can be overcome better by batch algorithms [48] than incremental algorithms, if the same number of samples are generated without adaptation to the environment. Inspired by the example, our enhanced RRT* algorithm will not discard failed samples immediately. Section 2.2 will describe the details on how to reuse failed samples in different ways from [48] or [40].

On the contrary, Figure 2-5 illustrates the behaviors by the RRT* and the FMT* in an open environment, assuming the same set of samples for both, but with a fortunate sequence of samples for the RRT*. Initial and goal samples (red) are located at upper-left and lower-right corners respectively, and a batch of samples (blue) are given. The RRT* receives rewarding samples at early iterations, and other subsequent samples do not contribute to better solutions. The FMT*, by its design, thoroughly sweeps the batch of samples before reaching the goal. In this example, the sweeping behavior becomes disadvantageous.

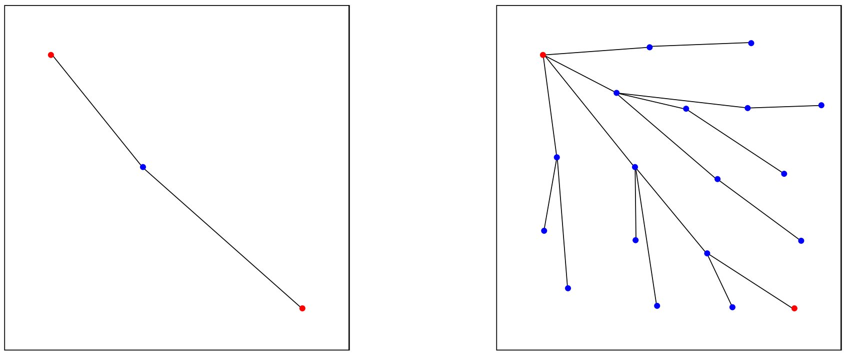

(a) RRT* (Good Case) (b) FMT*

Figure 2-5: The RRT* and the FMT* in an Open Environment

The example in Figure 2-5 leads to the discussion why incremental sampling-based algorithms, particularly RRTs, have been practical in high dimensions. As pointed out in [73], low-discrepancy and low-dispersion samples are keys to resolution complete algorithms, which become complete algorithms as the resolution gets sufficiently higher. However, resolution complete algorithms are directly connected to the curse of dimensionality in high dimensions and with fine discretization. Meanwhile, RRTs stimulate exploration toward unvisited regions and do not intend to sweep the space thoroughly, meaning that the best-case computation may not be subject to the curse of dimensionality issue whereas the worst-case computation still involves the issue. The RRT* efficiently improves the transient and asymptotic behaviors of RRTs, but remind that low-discrepancy and low-dispersion samples are assumed in the limit for the RRT*. Thus, the curse of dimensionality issue is inevitable to the RRT* as well.

Claim 2.9 (The curse of dimensionality) *We understand that the curse of dimensionality is inevitable in the long run and the transient behavior becomes more important for sampling-based algorithms, especially under uncertainties on computational budgets. In Section 2.2, we aim to propose an algorithm that finds the first solution quickly as in RRTs and switches its behavior toward optimal algorithms as in the RRT*, in incremental and anytime manners. Our enhanced RRT* algorithm in Section 2.2 never rejects long trajectory edges, which may happen more frequently at early iterations. In addition, we suppress algorithmic components such as rewiring* *operations and near neighbor queries until the first solution is found, as mentioned in Section 2.1.3.*

```
1 $V_n \leftarrow$ Vertex($z_{\text{root}}$); $E_n \leftarrow \emptyset$; $G_n \leftarrow (V_n, E_n)$; $V_{n,\text{goal}} \leftarrow \emptyset$;
2 **while** Interrupted() $=$ false **do**
3     $(z_{\text{samp}}, v_{\text{samp}}) \leftarrow$ Sample();
4     $V_{\text{near}} \leftarrow$ SelectNeighbors($z_{\text{samp}}$, backward_reachable);
5     $(v_{\text{new}}, e_{\text{new}}) \leftarrow$ ConnectToBestVertex($v_{\text{samp}}, V_{\text{near}}$);
6     $G_n \leftarrow (V_n, E_n, V_{n,\text{goal}}) \leftarrow$ UpdateData($v_{\text{new}}, e_{\text{new}}$);
7 **return** Traj($V_{n,\text{goal}}$) or Failure;
```

Algorithm 2.7: More Compact Form of Abstraction for the FMT*

```
   **Output**: $z_{\text{samp}}$ and $v_{\text{samp}}$
1  **if** Initialized() $=$ False **then**
2      $V_{n,\text{open}} \leftarrow V_{\text{n}}$; $V_{n,\text{closed}} \leftarrow \emptyset$; $V_{n,\text{remain}} \leftarrow$ SampleBatch();
3      $v_{n,\text{now}} \leftarrow \emptyset$; $V_{n,\text{ing}} \leftarrow \emptyset$;
4  **while** $V_{n,\text{ing}} = \emptyset$ **do**
5      $V_{n,\text{open}} \overset{-}{\leftarrow} v_{n,\text{now}}$; $V_{n,\text{closed}} \overset{+}{\leftarrow} v_{n,\text{now}}$; $v_{n,\text{now}} \leftarrow$ BestInCost($V_{n,\text{open}}$);
6      **if** $v_{n,\text{now}} = \emptyset$ or $v_{n,\text{now}} \in V_{n,\text{goal}}$ **then**
7          **return** Interrupt;
8      $V_{n,\text{ing}} \leftarrow V_{n,\text{remain}} \cap$ SelectNeighbors($v_{n,\text{now}}$, forward_reachable);
9  $V_{n,\text{ing}} \overset{-}{\leftarrow} (v_{\text{samp}} \in V_{n,\text{ing}})$; $z_{\text{samp}} \leftarrow$ State($v_{\text{samp}}$); $V_{n,\text{remain}} \overset{-}{\leftarrow} v_{\text{samp}}$;
10 **return** $(z_{\text{samp}}, v_{\text{samp}})$;
```

Algorithm 2.8: The `Sample` Procedure for the FMT*

```
   Input  : $v_{\text{in}}$, $V_{\text{near}}$, and $J_{\text{upper}}$ ($= \infty$ if unspecified)
   Output: $v$ and $e$
1  $v \leftarrow \emptyset$; $e \leftarrow \emptyset$;
2  for all $v_{\text{near}} \in V_{\text{near}} \cap V_{n,\text{open}}$ do
3      $(x_{\text{near}}, u_{\text{near}}, J_{\text{near}}) \leftarrow$ TPBVP(State($v_{\text{near}}$), State($v_{\text{in}}$));
4      if Cost($v_{\text{near}}$) $+ J_{\text{near}} < J_{\text{upper}}$ then
5          $(E_{\text{sort}}, J_{\text{sort}}) \overset{+}{\leftarrow}$ (Edge($v_{\text{near}}, v_{\text{in}}, x_{\text{near}}, u_{\text{near}}, J_{\text{near}}$), Cost($v_{\text{near}}$) $+ J_{\text{near}}$);
6  if ObstacleFree(Trajectory($e_{\text{sort}} \leftarrow$ BestInCost($E_{\text{sort}}$))) then
7      $v \leftarrow v_{\text{in}}$; $e \leftarrow e_{\text{sort}}$; $V_{n,\text{open}} \overset{+}{\leftarrow} v$;
8  else $V_{n,\text{remain}} \overset{+}{\leftarrow} v_{\text{in}}$ ;
9  return $(v, e)$;
```
Algorithm 2.9: The Laziest `ConnectToBestVertex` Procedure for the FMT*

Our representation of the FMT* algorithm (Algorithm 2.7, 2.8, and 2.9, with the `SelectNeighbors` procedure that returns $\mathcal{O}(\log n)$ near neighbors) reproduces the FMT* within our abstract form in Algorithm 2.1. Note that Algorithm 2.7 is a simplified form of Algorithm 2.1. As before, variables with subscripts n are treated as globally accessible data whereas variables with other explanatory subscripts remain local and temporary. Let V_n denote the union of $V_{n,\text{open}}$ and $V_{n,\text{closed}}$, and let $V_{n,\text{goal}}$ collect the set of vertices in the goal regions. To accommodate later algorithmic variations by simple modifications in `ConnectToBestVertex`, the `SelectNeighbors` procedure returns near neighbors in V_n, not in $V_{n,\text{open}}$. Compare that the FMT* finds near neighbors in $V_{n,\text{open}}$, but in our representation, the loop in Algorithm 2.9 considers the intersection of V_{near} and $V_{n,\text{open}}$ instead. Note that `PropagateInfo` in Algorithm 2.1 is simplified to `UpdateData` in Algorithm 2.7, to emphasize that no rewiring operations are needed for this batch algorithm.

In Algorithm 2.8, we pretend to provide a new sample, based on the batch of samples created during the initialization. The sampling pool $V_{n,\text{remain}}$, which initially stores the batch of samples, is depleted as the existing samples are drawn. The variable $v_{n,\text{now}}$ stands for the vertex with the best cost-to-come value among open vertices $V_{n,\,\text{open}}$ in the tree. Once all of samples $V_{n,\,\text{ing}}$ in neighborhood to $v_{n,\,\text{now}}$ are drawn, the vertex $v_{n,\,\text{now}}$ is closed. Note that the drawn sample may fail in connecting to the tree, then the sample is restored to $V_{n,\,\text{remain}}$ in Algorithm 2.9, but not to $V_{n,\,\text{ing}}$.

Algorithm 2.9 shows the laziest version of the `ConnectToBestVertex` procedure that only focuses on the best connection to nearby open vertices $V_{\text{near}} \cap V_{n,\,\text{open}}$. Compare the procedure with the basic version in Algorithm 2.2 and the lazy version in Algorithm 2.4, then note that this lazyness is the source for both the weaker convergence result mentioned in Remark 2.5 and the potential speed-up in execution [48]. The choice among several versions of `ConnectToBestVertex` is subject to the trade-off between the running time and the stronger convergence guarantees.

Proposition 2.10 (Equivalence of the FMT* [48] and our representation) *Let our representation of the FMT* be the composition of Algorithm 2.7, 2.8, and 2.9, with the* `SelectNeighbors` *procedure that returns $\mathcal{O}(\log n)$ near neighbors. Then, given the same batch of n random samples, the FMT* [48] and our representation of the FMT* are equivalent and generate the same solution.*

Proof (Sketch) Our representation of the FMT* is segmented to abstract actions `Sample`, `SelectNeighbors`, `ConnectToBestVertex`, and `UpdateData`, but produces the same algorithmic flow as the FMT* [48]. Thus, neglecting redundancies for segmented procedures, two algorithms are equivalent and generate the same result, given the same batch of samples. □

If the laziest computation in Algorithm 2.9 toward the best open candidate only is replaced by the lazy computation in Algorithm 2.4, our representation of the FMT* generates the same or better solution than the FMT*, by connecting to both open and closed vertices $V_n = V_{n,\,\text{open}} \cap V_{n,\,\text{closed}}$ in the tree. Then, Remark 2.5 extends to

$$c^* \leq \texttt{Cost}^{\text{PRM}^*}(V_{n,\,\text{goal}}) \leq \texttt{Cost}^{\text{our}}(V_{n,\,\text{goal}}) \leq \texttt{Cost}^{\text{FMT}^*}(V_{n,\,\text{goal}}).$$

Depending on the implementation choice for `ConnectToBestVertex`, our representation is equivalent to either the FMT* (with Algorithm 2.9) or the single-batch BIT*

(with Algorithm 2.4). Also, the trajectory cost by our algorithmic representation is lower bounded by the PRM*.

The choice of batch processing or incremental processing, i.e., an ordered sweep over candidate vertices or incremental inclusion of candidates into data structure, is subject to trade-offs depending on the circumstances. In order to import the behavior of ordered sweeping, incremental algorithms may choose to delay the propagation of updated information.

However, the maximal use of samples, especially the potential reuse of rare samples, is noticeable from the perspective of incremental algorithms, as motivated by the bug trap example in Figure 2-4. Algorithm 2.8 shows the intermediate step toward our enhanced RRT* to provide intuitions from batch algorithms. Before the presentation of our enhanced RRT* in Section 2.2, a different type of modification is considered in Section 2.1.6.

2.1.6 Shortcuts or Smoothing

Shortcuts or smoothing of jagged paths have been one of the canonical post-processing techniques for RRTs [74, 65]. The technique was repeated in the RRT* context [83, 52], but was attempted as algorithmic components to accelerate the convergence rate. In the RRT*-Smart [83], the `PathOptimization` procedure smooths the newly found trajectory if the cost is lower than the current best. In the process, beacons are defined and identified as centers for occasional biased sampling similar to Section 2.1.3. Later, the RRT*-Quick [52] augments the set of near neighbors with their ancestors up to the depth d, resulting in shortened or smoothed trajectories from the current sample's perspective. Benefits and worst-case losses by these modifications are rather obvious.

Remark 2.11 (Asymptotic optimality by shortcuts) *The RRT*-Smart [83] and the RRT*-Quick [52] are asymptotically optimal, i.e.,*

$$\mathbb{P}\left(\left\{\limsup_{i\to\infty} \mathtt{Cost}^{\mathrm{Smart}}(V_{i,\,\mathrm{goal}}) = c^*\right\}\right) = \mathbb{P}\left(\left\{\limsup_{i\to\infty} \mathtt{Cost}^{\mathrm{Quick}}(V_{i,\,\mathrm{goal}}) = c^*\right\}\right) = 1,$$

where c^* is the optimal cost to reach a state in goal regions from the initial state. Per iteration, shortcuts or smoothed trajectories are still considered as post-processed results, on top of normal operations by the RRT.*

Remark 2.12 (Further processing on trajectories) *Given the same sequence of samples during i iterations,*

$$c^* \leq \mathtt{Cost}^{\mathrm{RRT}^*-\mathrm{Smart}}(V_{i,\,\mathrm{goal}}) \leq \mathtt{Cost}^{\mathrm{RRT}^*}(V_{i,\,\mathrm{goal}})$$
$$c^* \leq \mathtt{Cost}^{\mathrm{RRT}^*-\mathrm{Quick}}(V_{i,\,\mathrm{goal}}) \leq \mathtt{Cost}^{\mathrm{RRT}^*}(V_{i,\,\mathrm{goal}}).$$

Note that inequalities hold at i-th iteration, not necessarily after spending the same amount of computation. Inequalities are straightforward results from the triangle inequality along trajectories

$$\mathtt{Cost}(a,b) + \mathtt{Cost}(b,c) \leq \mathtt{Cost}(a,c) \tag{2.4}$$

in metric spaces. The RRT-Quick augments the set of near vertices with ancestors up to the depth d, as a global operation. In other words, the RRT*-Quick considers at least $(d+1)$ multiples of $\mathcal{O}(\log n)$ vertices per iteration. The RRT*-Smart considers shortcuts along the best found trajectory only, as a local bias.*

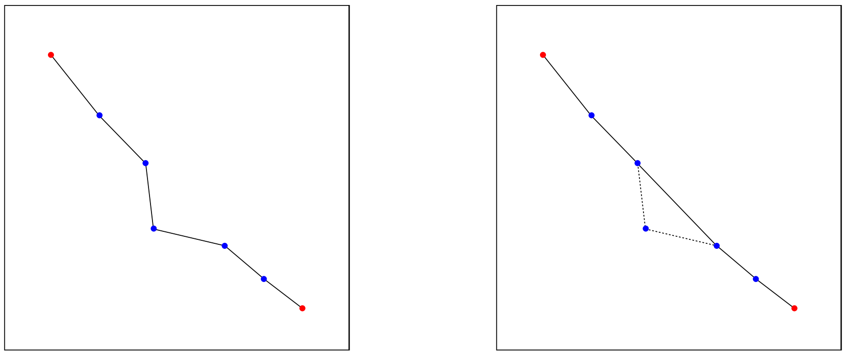

(a) Before Shortcuts (b) After Shortcuts

Figure 2-6: The Outcome of Shortcuts or Smoothing Operations

Figure 2-6 illustrates the outcome of smoothing along trajectories. The triangle inequality guarantees that the operation never generates worse solutions than before. However, in the worse-case scenario such as an extremely complicated maze, shortcuts may collide with obstacles with high probabilities. Therefore, it is easy to realize that additional computation for shortcuts is not always rewarding, depending on problem circumstances. Such procedures may remain as tunable operations, considering trade-offs between potential wastes and potential speed-ups.

The RRT*-Smart is reproduced within our abstraction, by slightly modifying `Sample`, `SelectNeighbors`, and `RewireTreeVertices` procedures. Note that revisiting vertices along a trajectory to goals is equivalent to smoothing the trajectory if the considered neighbors include ancestor vertices up to the specified depth d.

```
   Input  : $z_{\text{samp}}$, forward/backward, and the depth $d$
   Output: $V_{\text{near}}$
1  $V_{\text{near}} \leftarrow$ NearNeighbors($z_{\text{samp}}$, forward/backward, $V_i$);
2  for all $v_{\text{near}} \in V_{\text{near}}$ do
3      $v_{\text{now}} \leftarrow v_{\text{near}}$;
4      for $k = 2$; $k \leq d$; k++ do
5          $v_{\text{now}} \leftarrow$ Parent($v_{\text{now}}$); $V_{\text{near}} \overset{+}{\leftarrow} v_{\text{now}}$;
6  return $V_{\text{near}}$;
```
Algorithm 2.10: The `SelectNeighbors` Procedure for the RRT*-Smart

```
   Input  : $v_{\text{old}}$ and $v_{\text{rewire}}$
   Output: $V_{i+1}$ and $V_{i+1,\text{revisit}}$
1  $V_{i+1} \leftarrow$ UpdateDescendants($v_{\text{old}}, v_{\text{rewire}}$); $V_{i+1,\text{revisit}} \leftarrow V_{i,\text{revisit}}$;
2  if Cost($v \in V_{i+1,\text{goal}}$) $<$ Cost($v \in V_{i,\text{goal}}$) then
3      $v_{\text{now}} \leftarrow$ BestInCost($V_{i+1,\text{goal}}$);
4      while $v_{\text{now}} \neq \emptyset$ do
5          $V_{i+1,\text{revisit}} \overset{+}{\leftarrow} v_{\text{now}}$; $v_{\text{now}} \leftarrow$ Parent($v_{\text{now}}$);
6  return ($V_{i+1}, V_{i+1,\text{revisit}}$);
```
Algorithm 2.11: The `RewireTreeVertices` Procedure for the RRT*-Smart

Algorithm 2.10, 2.11, and 2.12 show one way of generalization for the RRT*-Smart within our abstraction Algorithm 2.1, where $d = 2$ is implied in the RRT*-Smart description [83]. Similar to the RRT$^\#$ in Section 2.1.4, the `RewireTreeVertices` procedure of Algorithm 2.3 fills in the globally accessible data $V_{i,\mathrm{revisit}}$ with vertices on the best found trajectory to goals. The difference from the RRT$^\#$ is that the RRT*-Smart adds vertices on the solution trajectory to $V_{i,\mathrm{revisit}}$, instead of non-descendant neighbors of the rewired vertex. The `Sample` procedure in Algorithm 2.12 assigns the revisit opportunity to the set $V_{i,\mathrm{revisit}}$ obtained in Algorithm 2.11, potentially resulting in shortcuts along the solution trajectory. The `SelectNeighbors` procedure in Algorithm 2.10 augments the set of $\mathcal{O}(\log n)$ near vertices with their ancestor vertices up to the depth d, thus revisits in the `Sample` procedure natively attempt to create shortcuts. As before, our representation of the RRT*-Smart is an intermediate step toward our enhanced RRT*, providing more intuitions than the practical implementation itself.

```
   Output: $z_{\mathrm{samp}}$ and $v_{\mathrm{samp}}$
1  if $V_{i,\mathrm{revisit}} \neq \emptyset$ then
2      $v_{\mathrm{samp}} \leftarrow$ PopTheBest($V_{i,\mathrm{revisit}}$); $z_{\mathrm{samp}} \leftarrow$ State($v_{\mathrm{samp}}$);
3  else if BiasToBeacons() $=$ true then
4      $z_{\mathrm{samp}} \leftarrow$ SampleInBeacons(); $v_{\mathrm{samp}} \leftarrow$ Vertex($z_{\mathrm{samp}}$);
5  else
6      $z_{\mathrm{samp}} \leftarrow$ SampleRandom(); $v_{\mathrm{samp}} \leftarrow$ Vertex($z_{\mathrm{samp}}$);
7  return $(z_{\mathrm{samp}}, v_{\mathrm{samp}})$;
```

Algorithm 2.12: The `Sample` Procedure for the RRT*-Smart

Proposition 2.13 (Equivalence with the RRT*-Smart [83]) *Let our representation of the RRT*-Smart be the composition of Algorithm 2.1, 2.2, 2.3, 2.10, 2.11, and 2.12 with the* `NearNeighbors` *procedure that returns $\mathcal{O}(\log n)$ near neighbors. Then, given the same sequence of random samples, our representation of the RRT*-Smart generates the same or better solution to the RRT*-Smart [83], with different schemes and amounts of computation.*

Proof (Sketch) The PathOptimization procedure in [83] finds shortcuts along the solution trajectory. Our maintenance and revisit to $V_{i,\text{revisit}}$ overgeneralize the RRT*-Smart's PathOptimization operation, due to the inflated output of the SelectNeighbors procedure by including ancestors of $\mathcal{O}(\log n)$ near vertices similar to the RRT*-Quick, i.e., more than direct ancestors of the inquired vertex only. Therefore, the obtained cost is at least the same to the RRT*-Smart, or better. □

Our SelectNeighbors procedure in Algorithm 2.10 considers the same set of $(d+1)\mathcal{O}(\log n)$ vertices to the RRT*-Quick, thus costlier than the RRT*'s $\mathcal{O}(\log n)$ neighbors. Our enhanced RRT* attempts to maintain the balance between the breadth-first inclusion (near neighbors) and the depth-first inclusion (direct ancestors) of vertices, within the $\mathcal{O}(\log n)$ complexity per iteration. Cost-to-come values of vertices and admissible heuristics for edge costs are key factors to such balancing.

2.2 Enhanced RRT* Algorithm

Through comparative literature reviews in Section 2.1, ideas for the enhanced RRT* algorithm have been enumerated to accelerate the RRT* within incremental $\mathcal{O}(\log n)$ operations per iteration. First, we revisit parts of vertices to propagate information over the underlying graph, not over the tree. Second, we reuse failed samples that are point-wise collision-free and promising in expected costs. Third, connection to ancestor vertices may shorten the trajectory. In addition, early iterations behave similar to RRTs by considering the nearest vertex only, before obtaining the first solution to goals. Canonical modifications in Section 2.1.3 are implied without explicit mentions. Algorithm 2.13 – 2.18 describe the enhanced RRT* algorithm. Most procedures are restated with minor changes from previous forms, while Algorithm 2.14 and 2.15 contain major changes, perhaps as generalized sampling and neighbor selection.

Compared to the previous form in Algorithm 2.1, the main loop of the enhanced RRT* in Algorithm 2.13 additionally requires a parameter d, the maximum depth for inclusion of ancestors or descendants. For convenience, the Sample only returns a vertex v_{samp}, and the SelectNeighbors takes the vertex v_{samp} as an input.

The `Sample` procedure in Algorithm 2.14 summarizes the generalized strategy to draw a sample for an iteration. Note that revisits and retries do not involve new randomized sampling. In this procedure, sampling in goals (Line 2), revisiting a vertex (Line 3), retrying a previously failed sample (Line 4), sampling around a solution trajectory (Line 6), and projecting a sample into promising regions (Line 9) are attempted with biases, while randomized sampling is the basic strategy (Line 8).

```
   Globally Accessible Data: $i$, $d$, $G_i$, $V_i$, $E_i$, $V_{i,\text{goal}}$, $V_{i,\text{revisit}}$, and $V_{i,\text{remain}}$
1  $i \leftarrow 0$; $V_i \leftarrow$ Vertex($z_{\text{root}}$); $E_i \leftarrow \emptyset$; $G_i \leftarrow (V_i, E_i)$; $V_{i,\text{goal}} \leftarrow \emptyset$;
2  while $i{+}{+} < N$ and Interrupted() $=$ false do
3      $v_{\text{samp}} \leftarrow$ Sample();
4      $V_{\text{near}} \leftarrow$ SelectNeighbors($v_{\text{samp}}$, backward_reachable);
5      $(v_{\text{new}}, e_{\text{new}}) \leftarrow$ ConnectToBestVertex($v_{\text{samp}}, V_{\text{near}}$);
6      $V_{\text{near}} \leftarrow$ SelectNeighbors($v_{\text{samp}}$, forward_reachable);
7      $G_{i+1} \leftarrow (V_{i+1}, E_{i+1}, V_{i+1,\text{goal}}) \leftarrow$ PropagateInfo($v_{\text{new}}, e_{\text{new}}, V_{\text{near}}$);
8  return Traj($V_{i,\text{goal}}$);
```

Algorithm 2.13: Main Loop of the Enhanced RRT*

```
   Output: $v_{\text{samp}}$
1  if $V_{i,\text{goal}} = \emptyset$ and BiasToGoals() $=$ true then
2      $z_{\text{samp}} \leftarrow$ SampleInGoals(); $v_{\text{samp}} \leftarrow$ Vertex($z_{\text{samp}}$);
3  else if BiasToRevisits() $=$ true then  $v_{\text{samp}} \leftarrow$ PopOne($V_{i,\text{revisit}}$) ;
4  else if BiasToReuse() $=$ true then  $v_{\text{samp}} \leftarrow$ PopOne($V_{i,\text{remain}}$) ;
5  else if $V_{i,\text{goal}} \neq \emptyset$ and BiasAroundTraj() $=$ true then
6      $z_{\text{samp}} \leftarrow$ SampleAroundTraj(Traj($V_{i,\text{goal}}$)); $v_{\text{samp}} \leftarrow$ Vertex($z_{\text{samp}}$);
7  else
8      $z_{\text{samp}} \leftarrow$ SampleRandom(); $v_{\text{samp}} \leftarrow$ Vertex($z_{\text{samp}}$);
9      if {Cost*($z_{\text{samp}}$) $+$ CostToGo*($z_{\text{samp}}$) $>$ BestInCost($V_{i,\text{goal}}$)} or
       {ObstacleFree($v_{\text{samp}}$)} then  $v_{\text{samp}} \leftarrow$ ProjectSample($v_{\text{samp}}$) ;
10 return $v_{\text{samp}}$;
```

Algorithm 2.14: The `Sample` Procedure for the Enhanced RRT*

```
Input  : $v_{\text{samp}}$ and fwd/bwd_reachable
Output: $V_{\text{near}}$
1  if $V_{i,\text{goal}} = \emptyset$ then
2      if bwd then  return $V_{\text{near}} \leftarrow \texttt{NearNeighbor}(v_{\text{samp}}, \text{bwd}, V_i)$ ;
3      else if fwd then  return $V_{\text{near}} \leftarrow \texttt{NearNeighbor}(v_{\text{samp}}, \text{fwd}, V_{i,\text{remain}})$ ;
4  else if $V_{i,\text{goal}} \neq \emptyset$ and bwd then
5      $V_{\text{cand}} \leftarrow \texttt{NearNeighbors}(v_{\text{samp}}, \text{bwd}, V_i)$; $l = |V_{\text{cand}} \cap \texttt{Ancestors}(v_{\text{samp}})|$;
6      for all $v_{\text{cand}} \in V_{\text{cand}} \setminus \texttt{Ancestors}(v_{\text{samp}})$ do
7          $k = 1$; $v_{\text{now}} \leftarrow v_{\text{cand}}$;
8          while $\{k = 1$ and $v_{\text{samp}} \in V_i$ and $\texttt{Timestamp}(v_{\text{now}}) < \texttt{Timestamp}(v_{\text{samp}})\}$
           or $\{k \leq d$ and $\texttt{Cost}(v_{\text{now}}) + \texttt{Cost}^*(v_{\text{now}}, v_{\text{samp}}) > \texttt{Cost}^*(v_{\text{samp}})\}$ do
9              $k{+}{+}$; $v_{\text{now}} \leftarrow \texttt{Parent}(v_{\text{now}})$;
10         if $k \leq d$ and $v_{\text{now}} \neq \emptyset$ then  $V_{\text{near}} \xleftarrow{+} v_{\text{now}}$ ;
11         else  $l{+}{+}$ ;
12     for $k = 1$; $k \leq l$; k++ do  $V_{\text{near}} \xleftarrow{+} \texttt{GrandParent}(v_{\text{samp}}, k+1)$ ;
13 else if $V_{i,\text{goal}} \neq \emptyset$ and fwd then
14     $V_{\text{cand}} \leftarrow \texttt{NearNeighbors}(v_{\text{samp}}, \text{fwd}, V_i \cup V_{i,\text{remain}})$;
15     $l = |V_{\text{cand}} \cap \texttt{Descendants}(v_{\text{samp}})|$;
16     for all $v_{\text{cand}} \in V_{\text{cand}} \setminus \texttt{Descendants}(v_{\text{samp}})$ do
17         $k = 1$; $v_{\text{now}} \leftarrow v_{\text{cand}}$;
18         while $\{k = 1$ and $v_{\text{samp}} \in V_i$ and $\texttt{Timestamp}(v_{\text{samp}}) < \texttt{Timestamp}(v_{\text{now}})\}$
           or $\{k \leq d$ and $\texttt{Cost}(v_{\text{samp}}) + \texttt{Cost}^*(v_{\text{samp}}, v_{\text{now}}) > \texttt{Cost}^*(v_{\text{now}})$ do
19             $k{+}{+}$; $v_{\text{now}} \leftarrow \texttt{RandomChild}(v_{\text{now}})$;
20         if $k \leq d$ and $v_{\text{now}} \neq \emptyset$ then  $V_{\text{near}} \xleftarrow{+} v_{\text{now}}$ ;
21         else  $l{+}{+}$ ;
22     for $k = 1$; $k \leq l$; k++ do
23         $v_{\text{now}} \leftarrow \texttt{RandomGrandChild}(v_{\text{samp}})$; $V_{\text{near}} \xleftarrow{+} v_{\text{now}}$;
24 return $V_{\text{near}}$;
```

Algorithm 2.15: The `SelectNeighbors` Procedure for the Enhanced RRT*

	backward_reachable	forward_reachable
With no solution	$\mathcal{O}(\log n)$ near, V_i	$\mathcal{O}(\log n)$ near, $V_{i,\,\text{remain}}$
With solutions	$\mathcal{O}(\log n)$ near, V_i modified with ancestors	$\mathcal{O}(\log n)$ near, $V_i \cup V_{i,\,\text{remain}}$ modified with descendants

Table 2.1: Outputs from the `SelectNeighbors` Procedure in Algorithm 2.15

The `SelectNeighbors` procedure in Algorithm 2.15 contains computation for four different combinations of inputs, depending on the existence of a solution trajectory and the backward/forward reachability from the sample of interest. Table 2.1 summarizes the outputs for each input combination. Before obtaining the first solution trajectory to goals, near neighbors are returned among tree vertices (backward reachable, Line 2) or previously failed samples (forward reachable, Line 3), to focus efforts on finding the first solution quicker than the RRT*. We do not bypass the `PropagateInfo` procedure in Algorithm 2.13, and instead, we try connections to previously failed samples $V_{i,\,\text{remain}}$ from the current sample v_{samp}. Once the first solution trajectory is obtained to goals, near neighbors are returned among tree vertices (backward reachable, Line 5) or the union of tree vertices and previously failed samples (forward reachable, Line 14). The set of near neighbors is further modified or augmented in the following lines. In Line 8 or 18, based on timestamps of vertices (if $k = 1$ and $v_{\text{samp}} \in V_i$), we can replace some of previously attempted vertices with their parents or children. Similarly, still in Line 8 or 18, lower bounds on expected costs (if $k \leq d$) can determine to replace unhelpful vertices with their parents or children. Such considerations are up to the depth d, and remaining quotas are assigned to direct ancestors or descendants of the vertex v_{samp}. This modification and augmentation of the set V_{near} maintain the balance between the breadth-first inclusion (ancestors/descendants of near neighbors) and the depth-first inclusion (direct ancestors/descendants), without increasing the $\mathcal{O}(\log n)$ complexity per iteration by a factor $(d + 1)$. Inclusion of ancestors or descendants in V_{near} leads to shortcuts of existing trajectories via the `ConnectToBestVertex` procedure.

The `ConnectToBestVertex` procedure in Algorithm 2.16 is similar to Algorithm 2.2, with its main difference in adding the failed collision-free sample v_{in} to the set $V_{i,\text{remain}}$. Note that the growth of the set $V_{i,\text{remain}}$ is discouraged by adding v_{in} in the laziest manner, i.e., by skipping to add any vertex v_{in} that can be safely connected to one of $\mathcal{O}(\log n)$ near neighbors in $V_{i,\text{remain}}$.

```
   Input  : $v_{\text{in}}$, $V_{\text{near}}$, and $J_{\text{upper}}$ ($= \infty$ if unspecified)
   Output: $v$ and $e$
1  $v \leftarrow \emptyset$; $e \leftarrow \emptyset$;
2  for all $v_{\text{near}} \in V_{\text{near}}$ do
3      $(x_{\text{near}}, u_{\text{near}}, J_{\text{near}}) \leftarrow$ TPBVP(State($v_{\text{near}}$), State($v_{\text{in}}$));
4      if Cost($v_{\text{near}}$) $+ J_{\text{near}} < J_{\text{upper}}$ and ObstacleFree($x_{\text{near}}$) then
5          $J_{\text{upper}} \leftarrow$ Cost($v_{\text{near}}$) $+ J_{\text{near}}$;
6          $v \leftarrow v_{\text{in}}$; $e \leftarrow$ Edge($v_{\text{near}}, v, x_{\text{near}}, u_{\text{near}}, J_{\text{near}}$);
7  if $v = \emptyset$ and $e = \emptyset$ and $v_{\text{in}} \notin V_i$ and ObstacleFree($v_{\text{in}}$) then
8      $V_{\text{near}} \leftarrow$ SelectNeighbors($v_{\text{samp}}$, forward_reachable, $V_{i,\text{remain}}$);
9      for all $v_{\text{near}} \in V_{\text{near}}$ do
10         $(x_{\text{near}}, u_{\text{near}}, J_{\text{near}}) \leftarrow$ TPBVP(State($v_{\text{in}}$), State($v_{\text{near}}$));
11         if ObstacleFree($x_{\text{near}}$) then return $(\emptyset, \emptyset)$ ;
12     $V_{i,\text{remain}} \overset{+}{\leftarrow} v_{\text{in}}$;
13 return $(v, e)$;
```

Algorithm 2.16: The `ConnectToBestVertex` Procedure for the Enhanced RRT*

The `PropagateInfo` procedure in Algorithm 2.17 is identical to Algorithm 2.3, whereas its sub-routine `RewireTreeVertices` in Algorithm 2.18 contains an integration of Algorithm 2.5 and 2.11. Instead of storing non-descendant neighbors to the rewired vertex, Line 2 stores the rewired vertex v_{rewire} to the set $V_{i,\text{revisit}}$. This modification embraces the previous computation in Algorithm 2.18 and incurs additional attempts for further refinement or shortcuts from the rewired vertex. Line 6 stores vertices along the improved trajectory to goals for revisits, expecting further refinement or shortcuts.

```
Input  : $v$, $e$, and $V_{\text{near}}$
Output: $V_{i+1}$, $E_{i+1}$, and $V_{i+1,\,\text{goal}}$
1 $V_{i+1} \leftarrow V_i \cup \{v\}$; $E_{i+1} \leftarrow E_i \cup \{e\}$; $V_{i+1,\,\text{goal}} \leftarrow V_{i,\,\text{goal}} \cup$ VertexInGoals$(v)$;
2 for all $v_{\text{near}} \in V_{\text{near}}$ do
3     $(v_{\text{rewire}}, e_{\text{rewire}}) \leftarrow$ ConnectToBestVertex$(v_{\text{near}}, \{v\},$ Cost$(v_{\text{near}}))$;
4     if $v_{\text{rewire}} \neq \emptyset$ and $e_{\text{rewire}} \neq \emptyset$ then
5         $E_{i+1} \overset{-}{\leftarrow}$ Edge(Parent$(v_{\text{near}}), v_{\text{near}})$; $E_{i+1} \overset{+}{\leftarrow} e_{\text{rewire}}$;
6         $V_{i+1} \leftarrow$ RewireTreeVertices$(v_{\text{near}}, v_{\text{rewire}})$;
7 return $(V_{i+1}, E_{i+1}, V_{i+1,\,\text{goal}})$;
```

Algorithm 2.17: The `PropagateInfo` Procedure for the enhanced RRT*

```
Input  : $v_{\text{old}}$ and $v_{\text{rewire}}$
Output: $V_{i+1}$ and $V_{i+1,\,\text{revisit}}$
1 $V_{i+1} \leftarrow$ UpdateDescendants$(v_{\text{old}}, v_{\text{rewire}})$; $V_{i+1,\,\text{revisit}} \leftarrow V_{i,\,\text{revisit}}$;
2 $V_{i+1,\,\text{revisit}} \overset{+}{\leftarrow} v_{\text{rewire}}$;
3 if Cost$(v \in V_{i+1,\,\text{goal}}) <$ Cost$(v \in V_{i,\,\text{goal}})$ then
4     $v_{\text{now}} \leftarrow$ BestInCost$(V_{i+1,\,\text{goal}})$;
5     while $v_{\text{now}} \neq \emptyset$ do
6         $V_{i+1,\,\text{revisit}} \overset{+}{\leftarrow} v_{\text{now}}$; $v_{\text{now}} \leftarrow$ Parent$(v_{\text{now}})$;
7 return $(V_{i+1}, V_{i+1,\,\text{revisit}})$;
```

Algorithm 2.18: The `RewireTreeVertices` Procedure for the Enhanced RRT*

Proposition 2.14 (Asymptotic optimality of the enhanced RRT*) *The enhanced RRT* (Algorithm 2.13 – 2.18) is asymptotically optimal, i.e.,*

$$\mathbb{P}\left(\left\{\limsup_{i\to\infty} \texttt{Cost}^{\text{Enhanced}-\text{RRT}^*}(V_{i,\,\text{goal}}) = c^*\right\}\right) = 1,$$

where c^* is the optimal cost to reach a state in goal regions from the initial state.

Proof The enhanced RRT* generalizes procedures of the RRT*, in such ways that the convergence guarantees and the $\mathcal{O}(\log n)$ complexity per iteration are not affected.

All of practical modifications enhance the best-case convergence rate to the optimum, while the worst-case performance is unchanged. $\square$

Remark 2.15 (Further processing per sample) *Given the same sequence of i randomized samples with no remaining element in $V_{i,\,\mathrm{revisit}}$,*

$$c^* \leq \mathtt{Cost}^{\mathrm{Enhanced-RRT}^*}(V_{j,\,\mathrm{goal}}) \leq \mathtt{Cost}^{\mathrm{RRT}^*}(V_{i,\,\mathrm{goal}}) \leq \mathtt{Cost}^{\mathrm{RRT}}(V_{i,\,\mathrm{goal}}).$$

Note that the inequality holds after processing i randomized samples, not necessarily after spending the same amount of computation. The different index $j \geq i$ for the enhanced RRT indicates the additional iterations triggered without randomly drawn samples, by revisiting vertices or retrying previously failed samples.*

The enhanced RRT* integrates ideas that are summarized, inspired, and adapted in Section 2.1, within incremental $\mathcal{O}(\log n)$ complexity per iteration. Due to trade-offs by modifications, certain algorithm may perform the best among compared algorithms in one environment, while performing the worst in another environment. Section 2.1 illustrates such scenarios, and our enhanced RRT* aims to avoid the worst by properly merging advantageous behaviors and by diluting disadvantageous behaviors. Some simulation experiments will follow in Section 2.5.

2.3 Feedback Planning Algorithm

In this section, we propose a tree-based feedback planning framework, GR-FMTs, as a simple but practical adaptation of the RRT*. GR-FMTs may be used for efficient replanning during the execution of planned trajectories, instead of generating feedback policies as a feedback planning algorithm. GR-FMTs are grounded on a goal-rooted backward RRT* structure with two main modifications: 1) on-demand feedback policy generation by connecting to the best vertex in the neighbor set, and 2) efficient mixture of lead-time, off-line, and on-line computation. Given solutions for optimal TPBVPs, GR-FMTs generate feedback policies that accomplish safe, dynamically feasible, and asymptotically optimal trajectories without resorting to discretization or interpolation schemes.

GR-FMTs consist of two phases: expansion (Algorithm 2.19) and execution (Algorithm 2.20), where the former phase grows the set of optimal motions off-line, in lead time, or during idle time between execution phases, and the latter phase runs on-line to generate feedback policies out of the available set of optimal motions.

In the previous setting, the procedures `Cost` and `CostToGo`* computed the cost-to-come value in a forward tree and the cost-to-go value estimate to goals. In the backward setting of GR-FMTs, let the `Cost` and `CostToCome`* procedures compute the cost-to-go value to goal vertices in a backward tree and the cost-to-come value estimate from the initial state.

2.3.1 GR-FMTs: Expansion Phase

```
Globally Accessible Data: $i$, $d$, $G_i$, $V_i$, $E_i$, $V_{i,\,\text{initial}}$, $V_{i,\,\text{revisit}}$, and $V_{i,\,\text{remain}}$
1 $i \leftarrow 0$; $(V_i, E_i) \leftarrow (\texttt{Vertex}(z_{\text{goal}}), \emptyset) \cup (\texttt{Vertex}(z \in Z_{\text{goal}}), E_{\text{goal}})$; $G_i \leftarrow (V_i, E_i)$;
2 $V_{i,\,\text{initial}} \leftarrow \emptyset$; $V_{i,\,\text{revisit}} \leftarrow \emptyset$; $V_{i,\,\text{remain}} \leftarrow \emptyset$;
3 while $i{+}{+} < N$ and $\texttt{Interrupted}() =$ false do
4     $v_{\text{samp}} \leftarrow \texttt{Sample}()$;
5     $V_{\text{near}} \leftarrow \texttt{SelectNeighbors}(v_{\text{samp}}, \text{forward\_reachable})$;
6     $(v_{\text{new}}, e_{\text{new}}) \leftarrow \texttt{ConnectToBestVertex}(v_{\text{samp}}, V_{\text{near}})$;
7     $V_{\text{near}} \leftarrow \texttt{SelectNeighbors}(v_{\text{samp}}, \text{backward\_reachable})$;
8     $G_{i+1} \leftarrow (V_{i+1}, E_{i+1}, V_{i+1,\,\text{initial}}) \leftarrow \texttt{PropagateInfo}(v_{\text{new}}, e_{\text{new}}, V_{\text{near}})$;
```

Algorithm 2.19: GR-FMTs (Expansion)

Algorithm 2.19 shows the main loop of GR-FMTs in the expansion phase, in a similar description to the RRT* (Algorithm 2.1) and the enhanced RRT* (Algorithm 2.13). Algorithm 2.19 differs from our previous representation in two aspects: 1) in populating Z_{goal}, a set of samples in goal regions and 2) in building trees rooted at goals. Assuming the state z_{goal} represents the goal, vertices out of Z_{goal} are inserted to trees and treated as virtual roots, where the edges to the parent are set to contain costs h (see (1.3)), i.e., penalty costs for final state discrepancy. This insertion of penalty costs natively introduces trade-offs in heading to an ultimate goal in the goal set.

Variables in the description of Algorithm 2.19 are similar to previous algorithms. Due to the backward setting, $V_{i,\text{initial}}$ replaces the role of $V_{i,\text{goal}}$, and the sampled vertex v_{samp} is connected to neighbors in the forward reachable set, instead of the backward reachable set. Note that some of the procedures need minor replacement of notations from previous forms, e.g., from $V_{i,\text{goal}}$ to $V_{i,\text{initial}}$.

The `Sample` procedure follows the `Sample` procedure for the enhanced RRT*, as in Algorithm 2.14. Accordingly, returned samples may depend on heuristics that are weighted higher for goals, existing verticies in the tree for revisits, previously failed samples for reuse, the initial state, or around the best found trajectory. The `SelectNeighbors` procedure may imply the enhanced `SelectNeighbors` procedure as in Algorithm 2.15 or the simple `NearNeighbors` procedure that returns $\mathcal{O}(\log n)$ near neighbors, depending on the user's preference. Due to fundamental difficulties with cost-to-go values as the pseudo-metric, small-time reachable sets $\mathcal{R}(z, t_f)$ or $\mathcal{R}(z, -t_f)$ with finite t_f may be considered as a superset of the $\mathcal{O}(\log n)$ vertices if the integrand g in (1.3) is always positive. In the absence of efficient algorithms to estimate actual costs or small-time reachable sets, using the Euclidean distance metric potentially degrades the performance due to failed steering attempts that are avoidable by other metrics. The `ConnectToBestVertex` procedure may imply any of previous descriptions in Algorithm 2.2, 2.4, 2.9, 2.16, 2.21, or 2.22. Similarly, the `PropagateInfo` procedure implies Algorithm 2.3 or 2.17.

Proposition 2.16 (Completeness and Optimality of GR-FMTs) *Seen from each vertex in trees toward goals, the expansion phase of GR-FMTs is probabilistically complete and asymptotically optimal as the number of vertices tends to infinity. Moreover, the expected computation time to find a feasible trajectory and the convergence rate to the optimum do not differ for forward and backward.*

Proof Note that proofs and guarantees in [57] are identical for the RRT* with forward trees and backward trees. □

2.3.2 GR-FMTs: Execution Phase

Algorithm 2.20 shows the triggered on-demand computation of feedback policies, based on GR-FMTs constructed off-line, in lead time, or during idle time between execution phases. The algorithm obtains feedback policies by means of steering toward $\mathcal{O}(\log n)$ near vertices, i.e., trying to minimize the overall future trajectory cost. The execution phase of GR-FMTs effectively exploits the system knowledge on TPBVP solutions. Contrary to interpolation-based approaches, GR-FMTs ignore contributions from nearby vertices that are unreachable within small time. More specifically, some vertices are rejected by conservative reachability checks, and other vertices fail to generate better trajectories than the trajectories to reachable neighbors.

```
1 $z \leftarrow$ CurrentState();
2 if $T_{\text{delay}} > 0$ then
3     $z \leftarrow$ Simulate($z, U_{\text{sim}}, T_{\text{delay}}$);
4 $V_{\text{near}} \leftarrow$ SelectNeighbors(Vertex($z$), forward_reachable);
5 $(v_{\text{new}}, e_{\text{new}}) \leftarrow$ ConnectToBestVertex(Vertex($z$), $V_{\text{near}} \cup \{v_{\text{last}}\}$);
6 $v_{\text{last}} \leftarrow$ Parent($v_{\text{new}}$);
7 $[u(0), u(T)] \leftarrow$ OutputControl($e_{\text{new}}, T$);
8 if $T_{\text{delay}} > 0$ then
9     $U_{\text{sim}} \overset{+}{\leftarrow} [u(0), u(T)]$;
```

Algorithm 2.20: GR-FMTs (Execution)

The `CurrentState` procedure observes or estimates the current state. If the actuation delay $T_{\text{delay}}>0$ is not negligible, the queue U_{sim} of delayed signals is simulated forward to predict a future state, at which control signals computed in the current iteration will be in effect. Line 6 stores the parent vertex v_{last} obtained by steering solution in Line 5, to inform the next iteration in Line 5. In a loop inside the `ConnectToBestVertex` procedure, this v_{last} is attempted first for the TPBVP, and if successful, computation for subsequent `TPBVP` procedures in the loop decreases by the updated upper bound J_{upper}. More details for the cost-informed TPBVP loop are explained in Section 2.4. Finally, first T portion of the best found solution is commanded in the `OutputControl` procedure, and is appended to the queue U_{sim} for later simulation if T_{delay} is not negligible. GR-FMTs generate outputs in a similar manner to MPC, but in an incremental and efficient way that entirely reuses the previous computation.

The execution phase of GR-FMTs inserts a vertex at the current state z, in effect. From z to goals, probabilistic completeness and asymptotic optimality still remain as the number of vertices in trees tends to infinity.

2.3.3 GR-FMTs: Efficient Replanning

For real-time applications, sampling-based algorithms often require replanning out of the previous trees at a high frequency, to account for dynamically changing environment, large deviations from the planned trajectory, emergency scenarios, etc. This idea of replanning has been reportedly successful and practical for the use and reuse of forward RRTs in an efficient manner [34, 67]. More recently, the work in [55] successfully demonstrated replanning using the RRT*. We note that more advance is avaiable by incorporating the backward algorithmic structure that is prevalent in dynamic programming [10, 13], optimal control [21], and artificial intelligence [29, 44, 106, 61].

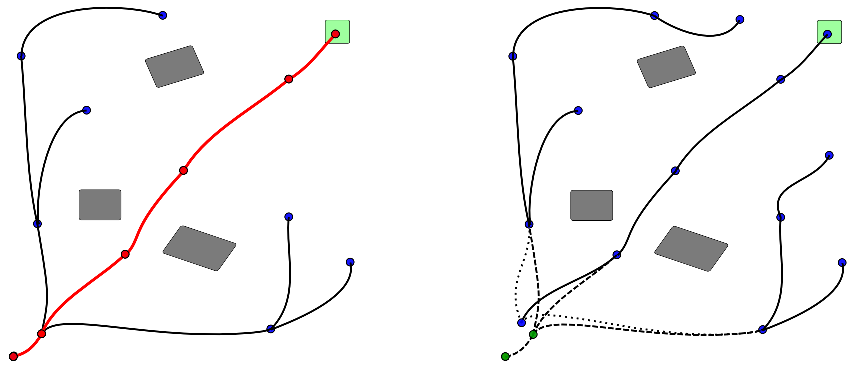

(a) Previous Iteration (b) Replanning

Figure 2-7: The Outcome of Replanning with Forward Algorithms

Figure 2-7 illustrates the replanning process with forward trees. In following a trajectory from the lower-left vertex to the goal region in the upper-right corner in

Figure 2-7a, the dynamical system may deviate from the planned trajectory significantly, to a degree the system cannot be stabilized over the planned trajectory. Then, new edges need to be constructed via TPBVPs from the deviated state to parts of the previous trees to reuse the previous trees as in Figure 2-7b, resulting in the loss of large branches due to failed TPBVP attempts with reasons such as collision, violation of constraints, limited reachability of the system, and so forth.

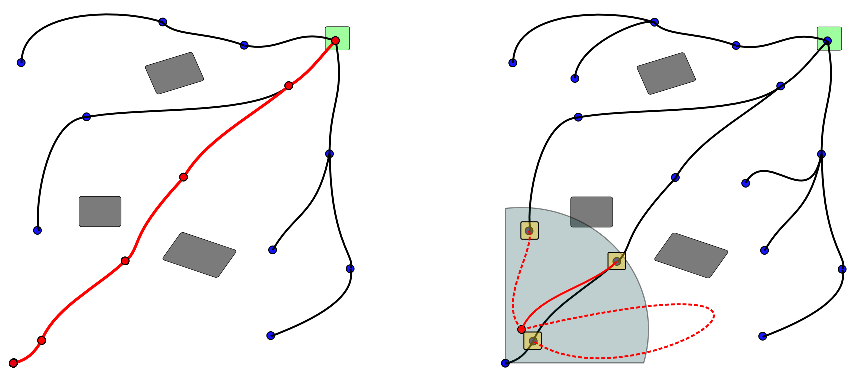

(a) Previous Iteration (b) Replanning

Figure 2-8: The Outcome of Replanning with GR-FMTs

On the other hand, Figure 2-8 illustrates the replanning process with GR-FMTs. To maintain the connectivity with the previous trees after a large deviation from the planned trajectory, the execution phase of GR-FMTs naturally attempts to connect to $\mathcal{O}(\log n)$ near neighbors in the backward trees. In effect, the replanning process with GR-FMTs (in fact, the execution phase in Algorithm 2.20) returns another trajectory from the deviated state in a principled way that minimizes the cost functional over the new trajectory, while effectively keeping most of the previous trees after replanning.

2.4 Cost-Informed TPBVPs for Dynamical Systems

As previously noted, dynamical systems induce expensive TPBVPs for sampling-based motion planning algorithms. For an affordable sampling-based algorithm with dynamical systems, we enhance the procedure `ConnectToBestVertex` into an efficient loop of cost-informed TPBVPs and collision checking as collective steering. Note that the previous descriptions in Algorithm 2.2, 2.4, 2.9, and 2.16 can be similarly enhanced by informing the TPBVP procedure about $J_{\text{upper}} - \texttt{Cost}(v_{\text{near}})$, i.e., the upper bound cost for the TPBVP, as in Line 3 of Algorithm 2.21 and 2.22. More specifically, a TPBVP solution (x_{near}, u_{near}, J_{near}) with

$$J_{\text{near}} > J_{\text{upper}} - \texttt{Cost}(v_{\text{near}}) \tag{2.5}$$

cannot contribute in connecting to the best vertex in V_{near}. Therefore, such a connection attempt can be rejected before computation or interrupted during the computation, without sacrificing the solution quality. Or often, the amount of computation required for the TPBVP is reduced by knowing the bound $J_{\text{upper}} - \texttt{Cost}(v_{\text{near}})$.

```
   Input  : $v_{\text{in}}$, $V_{\text{near}}$, and $J_{\text{upper}}$ ($= \infty$ if unspecified)
   Output: $v$ and $e$
1  $v \leftarrow \emptyset$; $e \leftarrow \emptyset$;
2  for all $v_{\text{near}} \in V_{\text{near}}$ do
3      $(x_{\text{near}}, u_{\text{near}}, J_{\text{near}}) \leftarrow \texttt{TPBVP}(\texttt{State}(v_{\text{in}}), \texttt{State}(v_{\text{near}}), J_{\text{upper}} - \texttt{Cost}(v_{\text{near}}))$;
4      if $\texttt{Cost}(v_{\text{near}}) + J_{\text{near}} < J_{\text{upper}}$ and $\texttt{ObstacleFree}(x_{\text{near}})$ then
5          $J_{\text{upper}} \leftarrow \texttt{Cost}(v_{\text{near}}) + J_{\text{near}}$;
6          $v \leftarrow v_{\text{in}}$; $e \leftarrow \texttt{Edge}(v_{\text{near}}, v, x_{\text{near}}, (u_{\text{near}}, J_{\text{near}}))$;
7  if $v = \emptyset$ and $e = \emptyset$ and $v_{\text{in}} \notin V_i$ and $\texttt{ObstacleFree}(v_{\text{in}})$ then
8      $V_{i,\text{remain}} \xleftarrow{+} v_{\text{in}}$;
9  return $(v, e)$;
```

Algorithm 2.21: The Cost-Informed ConnectToBestVertex Procedure (RRT*)

For clear and comparative representation, both Algorithm 2.21 for the RRT* (forward tree) and Algorithm 2.22 for GR-FMTs (backward tree) are shown, with minor notational differences in Line 3 and 6 only. The cost-informed loop of TPBVPs in Algorithm 2.21 and 2.22 differs from previous descriptions in Algorithm 2.2, 2.4, 2.9, and 2.16, only in Line 3. We pass the upper bound cost $J_{\text{upper}} - \texttt{Cost}(v_{\text{near}})$ in Line 3 for the TPBVP procedure, prevent unrewarding TPBVP attempts, and update the bound in Line 5 if a better connection is found.

```
Input  : $v_{\text{in}}$, $V_{\text{near}}$, and $J_{\text{upper}}$ ($= \infty$ if unspecified)
Output: $v$ and $e$
1 $v \leftarrow \emptyset$; $e \leftarrow \emptyset$;
2 for all $v_{\text{near}} \in V_{\text{near}}$ do
3     $(x_{\text{near}}, u_{\text{near}}, J_{\text{near}}) \leftarrow$ TPBVP(State($v_{\text{near}}$), State($v_{\text{in}}$), $J_{\text{upper}} -$ Cost($v_{\text{near}}$));
4     if Cost($v_{\text{near}}$) $+ J_{\text{near}} < J_{\text{upper}}$ and ObstacleFree($x_{\text{near}}$) then
5         $J_{\text{upper}} \leftarrow$ Cost($v_{\text{near}}$) $+ J_{\text{near}}$;
6         $v \leftarrow v_{\text{in}}$; $e \leftarrow$ Edge($v, v_{\text{near}}, x_{\text{near}}, u_{\text{near}}, J_{\text{near}}$);
7 if $v = \emptyset$ and $e = \emptyset$ and $v_{\text{in}} \notin V_i$ and ObstacleFree($v_{\text{in}}$) then
8     $V_{i,\text{remain}} \overset{+}{\leftarrow} v_{\text{in}}$;
9 return $(v, e)$;
```
Algorithm 2.22: The Cost-Informed `ConnectToBestVertex` Procedure (GR-FMTs)

Computational savings by cost-informed TPBVPs may happen in three ways. Due to difficulties in generalizing every possible type of TPBVP solutions, explanations are based on examples.

2.4.1 Rejection of Computation by the Informed Cost

For the actual cost J_{near} of the edge from v_{in} toward near vertices V_{near} in Line 3, a lower bound or an admissible heuristic is often available. For instance, a minimum-time or minimum-distance solution for Dubins' paths [31]

$$\begin{aligned} \dot{x} &= \cos\theta, \\ \dot{y} &= \sin\theta, \\ \dot{\theta} &= u, \quad -u_{\max} \le u \le u_{\max}, \end{aligned} \tag{2.6}$$

is shown to be one of the six possibilities in concatenating path segments: LRL, RLR, LSL, LSR, RSL, and RSR, where L denotes a left turn, R denotes a right turn, and S denotes a straight segment. Then, it is trivial that the length of Dubins' paths is lower bounded by the Euclidean distance between start and end states, i.e.,

$$\int_{t_0}^{t_f} \sqrt{\dot{x}(t)^2 + \dot{y}(t)^2}\, dt \geq \sqrt{(x(t_f) - x(t_0))^2 + (y(t_f) - y(t_0))^2}, \tag{2.7}$$

thus the Euclidean distance serves as an admissible heuristic for the actual trajectory length. Then, any connection that satisfies the inequality

$$\sqrt{(x(t_f) - x(t_0))^2 + (y(t_f) - y(t_0))^2} \geq J_{\text{upper}} - \texttt{Cost}(v_{\text{near}}) \tag{2.8}$$

cannot be a candidate for the best connection to near vertices. As a simple example, the length of the dotted line in Figure 2-9 is simple to compute as the Euclidean distance, and if the distance is longer than $J_{\text{upper}} - \texttt{Cost}(v_{\text{near}})$, such a connection can be rejected before computing six types of paths and finding the best out of six. Note that computational savings by rejection are enabled by informing the TPBVP about the upper bound cost, which is not a typical approach in TPBVPs.

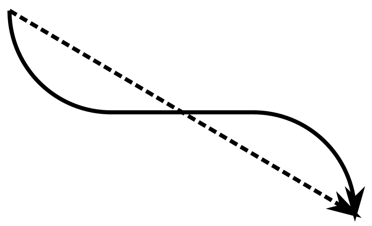


Figure 2-9: Admissible Heuristic for a Dubins' Path

2.4.2 Interruption of Computation by the Informed Cost

Computation of an optimal solution often combines several optimal solutions for parts. For instance, a minimum-time solution for two-dimensional double integrators

$$\begin{aligned} \ddot{x} &= u_x, \quad -u_{\max} \leq u_x \leq u_{\max}, \\ \ddot{y} &= u_y, \quad -u_{\max} \leq u_y \leq u_{\max}, \end{aligned} \tag{2.9}$$

first computes minimum-time solutions for each dimension, and later slows down the faster dimension to match the terminal time. Therefore, if the terminal time for any dimension is longer than $J_{\text{upper}} - \texttt{Cost}(v_{\text{near}})$, such a connection will surely have a terminal time longer than $J_{\text{upper}} - \texttt{Cost}(v_{\text{near}})$. For the 2D double integrators, there are two opportunities to interrupt the TPBVP.

Note that this interruption cannot be implemented without informing the TPBVP about the upper bound cost, which is not a typical approach in TPBVPs.

2.4.3 Reduction of Computation by the Informed Cost

TPBVP solvers often need to determine or search the terminal time for the trajectory. For instance, solvers in [117] or in Chapter 5 are subject to search for terminal time T in the range $T_{\min} \leq T \leq T_{\max}$. Thus, better specification of the upper bound cost $J_{\text{upper}} - \texttt{Cost}(v_{\text{near}})$ determines the upper bound $T_{\max}$ more tightly, and consequently reduces computational efforts for terminal time search in the range $T_{\min} \leq T \leq T_{\max}$.

2.5 Simulation Experiments

All computation results in this section are based on the processor Intel® Core™2 Extreme Q9300 @ 2.53GHz with 4GB RAM.

2.5.1 Enhanced RRT* Algorithm

As previously noted, it is sometimes not fair to directly compare two different algorithms with a large difference in implementation details. We attempt to compare some of our algorithmic components while constraining other parts as the same, in order to clearly show contributions by components. For that purpose, two million samples in the state space are randomly drawn and stored. Then, only the algorithmic components of interest are modified and compared using the stored random samples.

Branch and Bound

Section 2.1.3 presented the branch and bound technique in several forms. Figure 2-10 shows the effect and savings by the branch and bound technique, for the shortest path in a reasonably obstructed 2D environment. As a way of implementation for the branch and bound, informed sampling [39] in Figure 2-10b shrinks the region of sampling as an ellipse by the inequality (2.2). A more strict version of the branch and bound in Figure 2-10c considers the inequality (2.1) as well, in addition to the informed sampling. If the inequality (2.1) is considered alone without (2.2), note that the distribution of useful (yellow) and unrewarding (blue) samples would be different. Note that the visualization in Figure 2-10 is arranged to progressively strengthen the algorithmic component, to illustrate the order of savings by branch and bound.

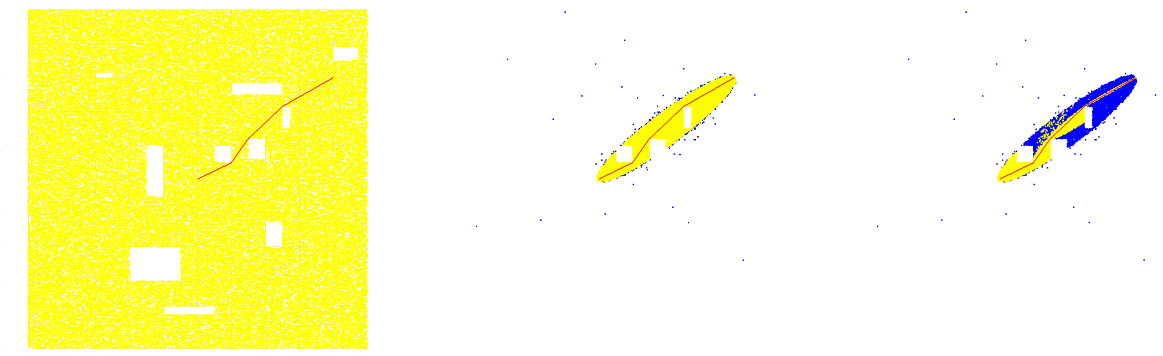
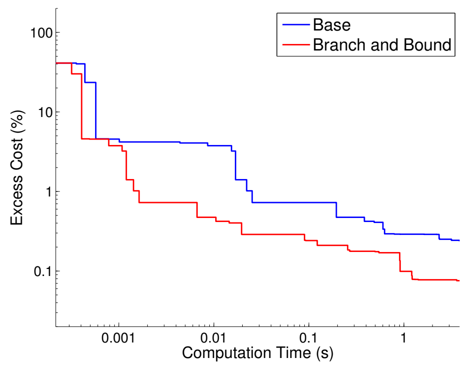
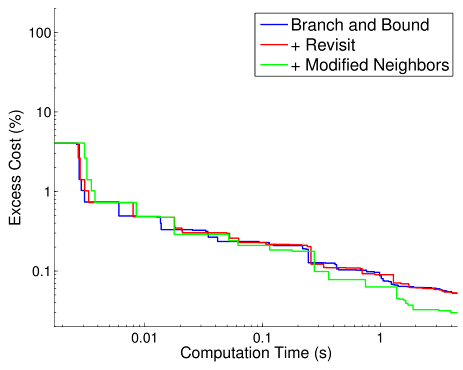

(a) Base (b) Informed Sampling [39] (c) Branch and Bound

Figure 2-10: The Effect and Savings by the Branch and Bound Technique (red: best path, white: obstacles or no samples, yellow: samples, blue: unrewarding samples)

Figure 2-11a, the comparison plot for the execution statistics, shows the contribution by the branch and bound technique (the mild version with the informed sampling only) in the same 2D environment to Figure 2-10. For better visualization, log scales are set for both x and y axes, where the excess cost percentage of the y axis is computed with respect to the actual shortest distance 10.163344 when moving a point mass from (0, 0) to (8, 6). The plot immediately shows that roughly 10 times of computation time is needed to achieve the same quality of solutions with the excess cost less than 5%. Such a specific number differs for various environmental setups and systems of interest, but an order of magnitude difference in computation time is noticeable from a practical point of view. All the subsequent experiments in this section will assume the branch and bound technique as the default component.



(a) Savings by the Branch and Bound



(b) Components of the Enhanced RRT*

Figure 2-11: The Effects of the Algorithmic Components in the Enhanced RRT*

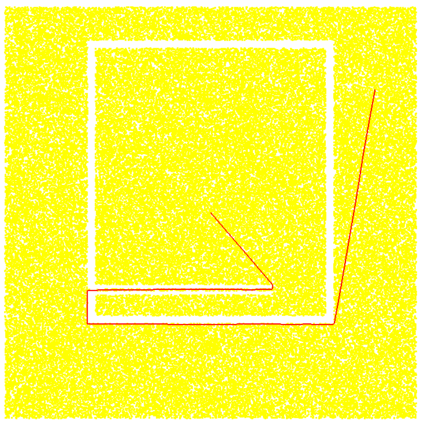
Revisit to Vertices and Modification of Near Neighbors

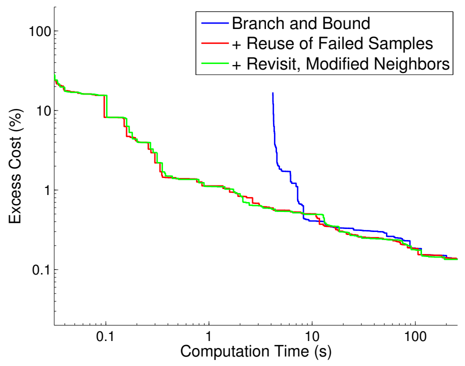
Two of the main components in the enhanced RRT* are compared in Figure 2-11b: 1) revisit to vertices as a graph-based propagation, and 2) modification for the set of near neighbors using ancestor and descendant vertices. Such components are added progressively, i.e., the blue line is by the branch and bound only, the red line combines the branch and bound technique and the revisit, and the green line contains the near neighbor modification in addition to the components for the red line. The magnitude of difference is certainly smaller than Figure 2-11a, but note that the potential room for improvement decreases as more algorithmic components are combined.

Per randomly drawn sample, better results are expected if more algorithmic components are combined in an inclusive manner. Therefore, the plot in Figure 2-11b is shown for computation time, instead of the number of randomly drawn samples. Roughly speaking from the plot, more algorithmic components incur some delays in the beginning, but long-term results get benefited.

Reuse of Failed Samples

As previously explained, the importance of rare samples, or the benefit of batch algorithms, becomes more explicit with the bugtrap example or the narrow corridor. In Figure 2-12, the remaining component of the enhanced RRT*, i.e., the reuse of failed samples, is finally included in the comparison experiment to handle the bugtrap example. Figure 2-12a follows the same convention to the previous Figure 2-10, i.e., red lines represent the best found path, white space implies obstacles or no samples, and yellow markers show the samples added to the trees. Note that the narrow corridor toward the outside space has a nearly invisible width of 0.1.


(a) Bugtrap Example (yellow: samples)



(b) Components of the Enhanced RRT*

Figure 2-12: The Bugtrap Example and the Components of the Enhanced RRT*

The excess cost percentage of the y axis in Figure 2-12b is assessed with respect to the actual shortest distance 38.984437 when moving a point mass outward from (0, 0) to (8, 6). The comparison plot immediately shows that the reuse of failed samples helps to find the first solution roughly 100 times faster or more. Including other components of the enhanced RRT* as well, the green line shows more improved results than the red line in longer terms.

All Components

Despite the danger of oversimplification, it is fair to summarize that the branch and bound technique is an overwhelming modification, i.e., the overhead for additional computation is negligible compared to the computational savings; the revisit to vertices and the modification of near neighbors are subject to trade-offs in results for shorter and longer terms; and the reuse of failed samples mostly affects the early phase until solutions with all homotopy classes are included in the trees.

2.5.2 Feedback Planning Algorithm

The expansion phase of GR-FMTs is not different from the RRT* or the enhanced RRT*, in terms of the computational complexity or execution statistics. The critical portion is the execution phase of GR-FMTs if real-time applications are concerned. Depending on the computation time, the execution phase may be used for replanning purposes at a low frequency or as a unified feedback planner at a high frequency. Examples in this chapter are considered rather trivial for this purpose, thus we borrow the example of torque-limited pendulum from Chapter 5.

Table 2.2 summarizes the statistics for simulated closed-loop executions using GR-FMTs as the feedback planner. With 0.1s of actuation delays and gaussian noises $\mathcal{N}(0, 0.01^2)$ for both θ and $\dot{\theta}$, all of 100 trials were able to reach and stabilize around the upright position. Since we lack efficient algorithms for searching $\mathcal{O}(\log n)$ near vertices using the pseudo-metric $\bar{J}$, we conservatively considered a larger number of vertices than necessary and saturated the number by 1500 or 500. Table 2.2 pro-

	$J = \int_0^{T_f} 1\,dt$ non-linear pendulum	$J = \int_0^{T_f} 1 + \ddot{\theta}^2\,dt$ linearized pendulum
Computation Time for Control Signal	8.24 ± 3.86 ms [2.76 ms, 37.88 ms]	14.72 ± 8.18 ms [4.62 ms, 44.89 ms]
Number of Nodes in Collective Steering	957.9 ± 189.5 [420, 1499]	382.9 ± 127.8 [5, 500]
Relative Cost	100.15 ± 0.23 % [100%, 104.16%]	100.01 ± 0.09 % [100%, 104.48%]

Table 2.2: Statistics for the execution phase of GR-FMTs (100 trials)

	$\mathcal{O}(\log n)$ neighbors by the Euclidean metric	$\mathcal{O}(\log n)$ neighbors by the actual cost
Rejection Dubins' Paths (Combinatorial)	0.4435 ± 0.1764	0.1584 ± 0.1165
Interruption 2D Double Integrators (Analytic)	0.5596 ± 0.1821	0.3415 ± 0.0909

Table 2.3: Computation Time (Relative Ratios) for the Cost-Informed TPBVPs

vides the relative cost to the case with unsaturated number of vertices. The statistics contains promising numbers for real-time applications, considering that several inefficiencies are present with less optimized codes.

2.5.3 Cost-Informed TPBVPs for Dynamical Systems

This section compares the normal loop of TPBVPs with the cost-informed TPBVPs, in a reasonably obstructed environment as in Figure 5-2. In fact, the existence of obstacles rarely affected this comparison result because the time was recorded only for the computation related to TPBVPs. The $\mathcal{O}(\log n)$ near neighbors were obtained by both the Euclidean metric and the cost-based pseudo-metric.

Rejection or Interruption of Computation by the Informed Cost

Table 2.3 summarizes the relative ratios for the computation time spent by the cost-informed TPBVPs. The relative ratios effectively correspond to the speed-up of 1.79 to 6.31 times, compared to the normal loop of TPBVPs. Notably, the efficiency gets more significant when the cost-based $\mathcal{O}(\log n)$ near neighbors can be obtained.

Reduction of Computation by the Informed Cost

Table 2.4 summarizes the relative ratios for the time horizon actually searched by the cost-informed TPBVPs. As promised, better information about $J_{\text{upper}} - \texttt{Cost}(v_{\text{near}})$, i.e., the upper bound cost for the TPBVP, leads to tighter bounds for terminal time search. Although the searched horizon does not directly imply the computation time, the factor of 15.6 to 53.8 is remarkable as the potential speed-up result. In this

	$\mathcal{O}(\log n)$ neighbors by the Euclidean metric	$\mathcal{O}(\log n)$ neighbors by the actual cost
Reduction 2D Double Integrators (Flatness)	0.0641 ± 0.0548	0.0186 ± 0.0161

Table 2.4: Searched Time Horizon (Relative Ratios) for the Cost-Informed TPBVPs

example, the TPBVP was solved by the flatness-based approach as in Chapter 5.

2.6 Conclusions

This chapter elaborated on our contributions to the framework of optimal sampling-based motion planning algorithms. Based on the comparative literature reviews within our algorithm abstraction, key factors of the state-of-the-art algorithms were identified and properly integrated as the enhanced RRT*, an incremental algorithm that improves the RRT*'s convergence rates strictly with the $\mathcal{O}(\log n)$ complexity per iteration. Next, as a sampling-based feedback planning algorithm or an efficient replanning method, GR-FMTs were proposed to generate collision-free, dynamically feasible, and asymptotically optimal feedback policies given the solutions for TPB-VPs. Lastly, the loop of cost-informed TPBVPs was designed to significantly reduce the computation time with dynamical systems, by means of rejection, interruption, or reduction of the expensive TPBVP computation.

Chapter 3

Numerical Local Steering for Nonlinear Systems

In this chapter, we provide a methodology for the steering problem of dynamical systems, described by nonlinear differential equations that involve high-dimensional state spaces. As an application example, time-optimal maneuvers for a high-speed off-road vehicle taking tight turns on a loose surface are studied using the RRT* algorithm. Our simulation results show that the aggressive skidding maneuver, usually called the trail-braking maneuver, naturally emerges from the RRT* algorithm as the minimum-time trajectory. In general, this integration of the RRT* and a TPBVP solver may serve as an anytime computation framework for nonlinear optimization problems in the presence of geometric constraints.

3.1 Introduction

There have been attempts to analyze and reproduce specialized human driving techniques, e.g., a minimum-time lane change, a minimum lap-time trajectory, a trail-braking maneuver, using optimal control theory [21], usually based on numerical optimization methods [46, 23, 115, 114]. Although earlier work focused on posing the problem as an optimization over a sequence of steady-state trim conditions, more recently, transient phases of extreme operating conditions were also taken into account using high-fidelity modeling [23].

Although these approaches are successful in describing and realizing certain aspects of the said vehicle maneuvers, the computation is usually carried out offline with careful transcription to numerical optimization formulations. Most algorithms of this class must be started with a feasible initial solution, which is usually hard to generate in the first place. Moreover, some numerical optimization methods suffer from local optimality, except for few unrealistic problem instances. While handling of differential constraints is efficient in most nonlinear programming methods (e.g., in a collocation-based algorithm), imposing geometrical constraints in configuration space, e.g., road boundaries, turns out to be challenging [58]. In this chapter, we use motion planning methods to generate optimal trajectories for minimum-time maneuvering of high-speed off-road vehicles.

This chapter focuses on using the RRT* algorithm to solve the optimal motion planning with probabilistic guarantees for complex dynamical systems with high-dimensional state spaces. Systems of this nature include those that have high maneuvering capabilities, such as race cars and aerobatic airplanes. Earlier work in [56] had assumed that the existence of a "steering" function that can find a control input that exactly connects an initial state to a final state, which is usually non-trivial to construct for such systems. Our first contribution is to extend the RRT* algorithm by relaxing this assumption to allow "approximate" steering functions. Second, we interpret the RRT* as an anytime computation framework for optimization problems with complex differential and geometric constraints. Third, using the resulting algorithm, we numerically analyze time-optimal maneuvers for a high-speed off-road vehicle.

This chapter is organized as follows. In Section 3.2, several extensions of the RRT* algorithm are introduced to handle systems with differential constraints and high-dimensional state spaces. Implementation details for an off-road rally car dynamics are provided in Section 3.3, and simulation results are discussed in Section 3.4. The chapter concludes in Section 3.5.

3.2 Modifications to the RRT*

In this chapter, the original version of the RRT* [57] serves as the baseline algorithm. Compared to our representation of the basic RRT* in Chapter 2 (Algorithm 2.1, 2.2, and 2.3 with the `SelectNeighbors` procedure that returns $\mathcal{O}(\log n)$ near neighbors), this version constructs the preliminary edge from the nearest vertex to the sample before selecting and connecting to near vertices. This section discusses several modifications to the RRT* algorithm to allow planning for systems with complex differential constraints involving high-dimensional state spaces.

3.2.1 Task Space Planning

To effectively deal with high-dimensional state spaces, the RRT* algorithm operates in a task space, which has a smaller dimension when compared to the state space $X \subset \mathbb{R}^n$ (see [103] for a discussion on task spaces). More precisely, let $T \subset \mathbb{R}^{n'}$ be a compact set. Recall that the set of inputs $U \subset \mathbb{R}^m$ is a subset of the m-dimensional Euclidean space. Usually, n' is much smaller than n. For practical purposes, it is tacitly assumed that there exists a surjective mapping $\mathcal{T} : X_{\text{free}} \to T$ that maps each collision-free state to its equivalent in the task space. Moreover, the `Sample` procedure returns i.i.d. random samples from the task space, and the steering procedure operates in the task space as explained below.

3.2.2 Steering Procedure

In the description of the RRT* algorithm, it is assumed that for any given $z_1, z_2 \in X_{\text{free}}$, the `TPBVP`$(z_1, z_2)$ procedure returns an *optimal* trajectory that starts from z_1 and reaches z_2 exactly, when such a trajectory exists. Finding a trajectory that connects z_1 and z_2 in this manner may be computationally challenging, since it amounts to solving a two-point boundary value problem for an ordinary differential equation. For certain dynamical systems, e.g., single integrator, a double integrator, or a curvature-constrained car (i.e., Dubins' vehicle) [31], analytic solutions to this boundary value problem do exist [31, 13]. However, an analytic solution to this problem is not available for most dynamical systems [20].

In what follows, we provide the implementation details of a steering procedure that is approximate in the following sense: the trajectory $x : [0, T] \rightarrow X_{\text{free}}$ generated by the $\texttt{TPBVP}(z_1, z_2)$ procedure is such that (i) x starts at z_1, i.e., $x(0) = z_1$, (ii) reaches a neighborhood of z_2 in the task space, i.e., there exists some $\delta \geq 0$ such that $\mathcal{T}(x(T)) \in B(\mathcal{T}(z_2); \delta)$[1], and (iii) has cost $c^* + \epsilon$, where c^* is the cost of the optimal trajectory that starts from z_1 and reaches z_2. Here, the parameters δ and ϵ are bounds on the connection error and sub-optimality of the trajectory, respectively.

Our steering function is based on numerical methods for solving differential equations [93] described in detail below.

Piecewise-constant Input

The steering function considers only constant inputs. More precisely, the trajectory $x : [0, T] \rightarrow X_{\text{free}}$ returned by $\texttt{TPBVP}(z_1, z_2)$ is such that $\dot{x} = f(x(t), \bar{u})$ and $x(0) = z_1$ for some $\bar{u} \in U$.

Shooting Method

The steering procedure proposed in this chapter is based on the shooting method [93], which can be used with either the bisection method or the false position method in order to determine the constant input value $\bar{u}$ for local steering. If the method does not converge within a given number of iterations, the connection is regarded as infeasible.

Before providing the shooting method, let us note the following definitions. Given a constant input $\bar{u} \in U$ and an initial state $\bar{z} \in X$, let $x(t; \bar{z}, \bar{u})$ denote the resulting trajectory of the dynamical system when started from initial state $\bar{z}$ under constant input $\bar{u}$, i.e., $\dot{x}(t; \bar{z}, \bar{u}) = f(x(t; \bar{z}, \bar{u}), \bar{u})$ for all $t \in [0, \infty)$ and $x(0, \bar{z}, \bar{u}) = \bar{z}$. Given some $\bar{t} \in \mathbb{R}_{\geq 0}$, let $x([0, \bar{t}]; \bar{z}, \bar{u})$ denote the restriction of the trajectory $x(\cdot; \bar{z}, \bar{u})$ to the interval $[0, \bar{t}]$. Let $\texttt{Term} : T \rightarrow \{\texttt{false}, \texttt{true}\}$ denote a termination condition that associates a terminal condition with each task space state. Finally, let $\texttt{Bisect}$:

[1]In the sequel, for a subset A of the d-dimensional Euclidean space, the set $B(a, r) \subset A$ denotes the closed ball of radius $r \in \mathbb{R}_{\geq 0}$ centered at $a \in A$, i.e., $B(a, r) := \{a' \in A \mid \|a' - a\| \leq r\}$.

$(z_1, z_2, z_3, u) \to u'$ be a bisection algorithm that returns an increment for the input so as to steer the terminal state of the dynamical system towards z_2, when the dynamical system is started at state z_1 and the current constant input u under consideration steers the system to state z_3.

```
1 $\bar{u} \leftarrow 0$;
2 for $i = 1$ to $M$ do
3     $\bar{t} \leftarrow \inf_{t \in [0,\infty)} \{t \mid x(t; z_1, \bar{u}) \in \mathrm{Term}(\mathcal{T}(z_2))\}$;
4     $\bar{z} \leftarrow x(\bar{t}; z_1, \bar{u})$;
5     if $\mathcal{T}(\bar{z}) \in B(\mathcal{T}(z_2); \delta)$ then
6         return $x([0, \bar{t}]; z_1, \bar{u})$;
7     else
8         $\bar{u} \leftarrow \bar{u} + \texttt{Bisect}(z_1, z_2, \bar{z}, \bar{u})$;
```

Algorithm 3.1: The TPBVP(z_1, z_2) procedure based on shooting

The shooting method is given in Algorithm 3.1. The algorithm initially starts with the zero input (Line 1). The state where the trajectory of the system starting from z_1 enters Term$(\mathcal{T}(z_2))$, i.e., the terminal set associated with z_2, is determined in Lines 3-4. This state is denoted as $\bar{z}$. If $\bar{z}$ is within the δ-neighborhood of $\mathcal{T}(z_2)$, then the algorithm returns the trajectory that reaches $\bar{z}$ (Line 6). Otherwise, a new input is selected using the Bisect function (Line 8), and the procedure is continued until the maximum number of iterations (denoted as M in Algorithm 3.1) is reached.

Repropagation

Since the steering function is approximate, i.e., can only reach a neighborhood of the final state, the rewiring procedure of the RRT* cannot be implemented directly as errors induced by the steering function in the rewiring step causes inconsistency in the states of descendant nodes. To handle this issue, corresponding descendant nodes are re-simulated using the stored sequences of inputs.

Before presenting the repropagation procedure, let us note the following definitions. Recall that (V, E) denotes the graph maintained by the RRT* algorithm. Given a state $z \in V$ in the tree, let `Children`(z) denote the set of all children vertices of z, i.e., `Children`$(z) := \{z' \in V \,|\, (z, z') \in E\}$. Given an edge $e = (z_1, z_2) \in E$, let `Input`(e) denote the input that drives the dynamical system from the state z_1 to the state z_2, and `Time`(e) denote the time it takes for this trajectory to reach z_2 starting from z_1. Clearly, `Children`, `Input`, and `Time` functions can be populated incrementally as the RRT* algorithm proceeds. Hence, these functions can be evaluated quickly, e.g., without re-running the `Steer` procedure.

The repropagation procedure is given in Algorithm 3.2, called whenever the extension towards an existing node $z \in V$ reaches a deviated state $\bar{z} \neq z$. This procedure recursively calculates the new state, denoted by z_{new}, for all the descendants of z.

```
1 for all $z' \in$ Children($z$) do
2     $z'_{\text{new}} \leftarrow x(\texttt{Time}(\texttt{Edge}(z, z')); \bar{z}, \texttt{Input}(\texttt{Edge}(z, z')))$;
3     Repropagate($z'$, $z'_{\text{new}}$);
4 $V \leftarrow V \setminus \{z\}$;
5 $V \leftarrow V \cup \{z_{\text{new}}\}$;
```

Algorithm 3.2: The `Repropagate`$(z, \bar{z})$ procedure

Clearly the repropagation may render some marginally-safe trajectories collide with adjacent obstacles. However, we have observed in experiments that the tree quickly recovers the lost edges with better trajectories.

3.2.3 Conditional Activation of the RRT*

In order to find a feasible solution quickly, we run the RRT algorithm until the algorithm returns a feasible solution. Once a feasible solution is obtained, the RRT* algorithm is run as is. More precisely, until a feasible solution is found, the `SelectVertices` procedure returns the nearest vertex only. Note that this modification is one of the algorithmic components of the enhanced RRT* in Chapter 2.

3.2.4 Branch-and-Bound

As mentioned in Section 2.1.3, application of the branch-and-bound algorithm [68] in graph search algorithms reduces the size of data structure and consequently the required computation [99, 67, 55]. In general, a cost-to-go function close to the optimal cost is computationally expensive to get. However, in practice, even loose approximations contribute to significant pruning of the search tree. We assume that a cost-to-go function is available and the $\texttt{CostToGo}(z)$ procedure implements the computation. Once a solution and its end state x_{soln} in the goal region are available, then any vertex z with $\texttt{Cost}(z) + \texttt{CostToGo}(z) > J(x_{soln})$ can be removed, without degrading the solution quality.

3.2.5 Reachability

Systems subject to non-honomic differential constraints and input saturation have smaller reachable sets (e.g., in terms of Lebesgue measure) when compared to holonomic dynamical systems. Motivated by the assumption that sampling is relatively cheaper than edge expansion, reachability information had been used to improve the RRT algorithm [104]. Clearly, the RRT* attempts to expand more edges when compared to the RRT. Moreover, the cost of edge expansion becomes even larger with higher fidelity trajectory simulation. Therefore, the knowledge of the reachable set must be useful for the RRT*, especially for systems with a high-dimensional state space. In general, computing reachable sets exactly is known to be computationally challenging [7]. However, system-specific approximations thereof usually can be computed rather easily.

3.3 Application to High-speed Off-road Vehicles

In this section, a nonlinear half-car vehicle dynamics is considered to simulate a rally car driving on loose surface. Specifically, a set of cornering maneuvers for various road curvatures are generated. The half-car model is less general than a full-body dynamics with suspensions, but it sufficiently captures the longitudinal load transfer phenomenon that takes an important role in cornering maneuvers [115].

3.3.1 Vehicle Dynamics

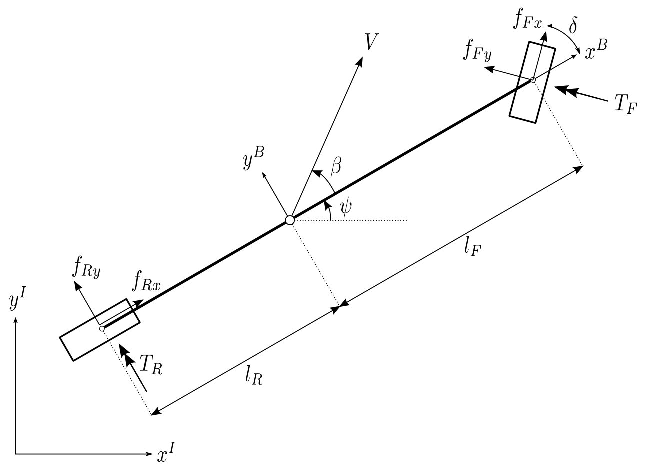


Figure 3-1: The Half-Car Vehicle Model

Let m and I_z denote the mass and the inertia of the vehicle. Let I_i, r_i, and ω_i ($i \in \{F, R\}$) denote the moment of inertia, the tire radius, and the angular velocity, respectively, where the subscripts F and R denote the front or rear wheels. Let x and y denote the position of the vehicle's center of gravity and ψ denote the yaw angle, in the inertial reference frame. Let V denote the speed of the vehicle. Let f_{Fx} and f_{Fy} denote the longitudinal and lateral forces acting on the front wheel, and f_{Rx} and f_{Ry} denote those acting on the rear wheel. Let β denote the side-slip angle. Let δ and T_i ($i \in \{F, R\}$) denote the steering angle and the torque acting on each wheel, respectively. See Figure 3-1 for a depiction of these variables.

Then, the equations of motion can be written as follows:

$$m\ddot{x} = f_{Fx}\cos(\psi+\delta) - f_{Fy}\sin(\psi+\delta) + f_{Rx}\cos\psi - f_{Ry}\sin\psi, \tag{3.1}$$

$$m\ddot{y} = f_{Fx}\sin(\psi+\delta) + f_{Fy}\cos(\psi+\delta) + f_{Rx}\sin\psi + f_{Ry}\cos\psi, \tag{3.2}$$

$$I_z\ddot{\psi} = (f_{Fy}\cos\delta + f_{Fx}\sin\delta)l_F - f_{Ry}l_R, \tag{3.3}$$

$$I_F\dot{\omega}_F = T_F - f_{Fx}r_F, \quad I_R\dot{\omega}_R = T_R - f_{Rx}r_R. \tag{3.4}$$

The tire force f_{ij} ($i \in \{F, R\}$, $j \in \{x, y\}$) depends on the normal force f_{iz} ($i \in \{F, R\}$) and the friction coefficient μ_{ij} ($i \in \{F, R\}$, $j \in \{x, y\}$) determined by Pacejka's Magic Formula [8]. The equations are

$$f_{ij} = \mu_{ij} f_{iz}, \quad (i = F, R,\ j = x, y); \tag{3.5}$$

$$\mu_{ij} = -\frac{s_{ij}}{s_i}\mu_i(s_i), \quad (i = F, R,\ j = x, y); \tag{3.6}$$

$$\mu_i(s_i) = D_i \sin(C_i \tan^{-1}(B_i s_i)), \quad (i = F, R). \tag{3.7}$$

The slip ratio, denoted by s_i, is calculated as follows:

$$s_i = \sqrt{s_{ix}^2 + s_{iy}^2}, \quad (i = F, R); \tag{3.8}$$

$$s_{ix} = \frac{V_{ix} - \omega_i r_i}{\omega_i r_i}, \quad (i = F, R) \tag{3.9}$$

$$s_{iy} = \frac{V_{iy}}{\omega_i r_i}, \quad (i = F, R). \tag{3.10}$$

The velocities at the wheels are calculated by

$$V = \sqrt{\dot{x}^2 + \dot{y}^2}, \tag{3.11}$$

$$\beta = \tan^{-1}\left(\frac{\dot{y}}{\dot{x}}\right) - \psi, \tag{3.12}$$

$$V_{Fx} = V\cos(\beta - \delta) + \dot{\psi} l_F \sin\delta, \tag{3.13}$$

$$V_{Fy} = V\sin(\beta - \delta) + \dot{\psi} l_F \cos\delta, \tag{3.14}$$

$$V_{Rx} = V\cos\beta, \tag{3.15}$$

$$V_{Ry} = V\sin\beta - \dot{\psi} l_R. \tag{3.16}$$

Finally, on a flat surface, normal forces are calculated by

$$f_{Fz} = \frac{l_R mg - hmg\mu_{Rx}}{(l_F + l_R) + h(\mu_{Fx}\cos\delta - \mu_{Fy}\sin\delta - \mu_{Rx})} \tag{3.17}$$

$$f_{Rz} = mg - f_{Fz} \tag{3.18}$$

where h is the height of the vehicle's center of gravity.

We assume that both input signals, namely the steering angle δ and the engine/brake torque T_i ($i \in \{F, R\}$), are limited to ranges, i.e., $\delta \in [\delta_{\min}, \delta_{\max}]$ and $T_i \in [T_{i,\min}, T_{i,\max}]$. Vehicle parameters are set identical to those given in [115].

3.3.2 Implementation details

In this subsection, several technical details are described in applying the RRT* algorithm and its extensions to the nonlinear half-car vehicle dynamics.

Cost Functional

As the cost functional to be minimized for each trajectory, the total travel time is used since we aim to acquire the minimum-time cornering maneuver. Edges in the tree are constructed by forward simulation of the dynamics with time steps 0.005s.

Sampling Strategy

The full state space consists of 8 variables x, y, $\dot{x}$, $\dot{y}$, ψ, $\dot{\psi}$, ω_F, and ω_R. Our sampling happens uniformly at a 4-dimensional task space (x, y, V, ψ) to enable the false position method with piecewise-constant inputs. The position (x, y) is sampled on free space, and the velocity and the yaw angle are sampled within ranges $V \in [V_{min}, V_{max}]$ and $\psi \in [\psi_{min}, \psi_{max}]$. The yaw angle ψ_t of the road tangential guides the ψ range such that $\psi \in [\psi_t - \Delta, \psi_t + \Delta]$ where Δ is chosen sufficiently large.

Distance Metric

In searching the nearest neighbor or nearby vertices within a ball, a metric is necessary for distance evaluation between two vertices. We define the distance metric as the Euclidean distance divided by the average speed. More precisely, given two states $z_i = (x_i, y_i, \dot{x}_i, \dot{y}_i, \psi_i, \dot{\psi}_i, \omega_{Fi}, \omega_{Ri}) \in X$ for $i \in \{1, 2\}$, the distance function is computed as

$$\mathrm{d}(z_1, z_2) = \frac{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}{(\sqrt{\dot{x}_1^2 + \dot{y}_1^2} + \sqrt{\dot{x}_2^2 + \dot{y}_2^2})/2}. \tag{3.19}$$

The choice is aligned with the discussion in [74] that the optimal cost-to-go is likely to be a good choice.

Reachability

Exact estimation of the reachable set is a computationally intensive task. However, most often, computationally-efficient conservative estimates of the reachable set of a dynamical system are available. In the sequel, an estimate of the reachable set is said to be conservative if it includes the reachable set itself. For a more precise definition, let $T \in \mathbb{R}_{>0}$ and $z \in X$. Given a dynamical system of the form in Equation (3.1)-(3.4), an initial state z, and a control input $u \in \mathcal{U}$, recall that $x(t; z, u)$ denotes the unique solution of the differential equation with initial state z and input u. Then, the T-reachable set of a dynamical system described by this differential equation is defined as the set of all states that are reachable from z by some admissible control input before time T, i.e., $\mathcal{R}_T(z) = \{z' \in Z \mid \exists u \in \mathcal{U}, t \in [0, T] \text{ such that } x(t; z, u) = z'\}$.

Below, we provide a procedure for computing a conservative estimate $\hat{\mathcal{R}}_T(z)$ of the reachable set of the dynamical system described in Section 3.3.1. This estimate assumes that bounds on the acceleration of the car as well as the yaw rate are known, and denoted by $\dot{V}_{\max}$ and $\dot{\psi}_{\max}$, respectively. For notational convenience, we describe computing the complement of $\hat{\mathcal{R}}_T(z)$. Given two states $z_1, z_2 \in X$, we have $z_2 \notin \hat{\mathcal{R}}_t(z_1)$ whenever at least one of the following holds:

- average yaw rate is greater than the maximum allowed, i.e.,

$$\left| \frac{\psi_2 - \psi_1}{\mathrm{d}(z_1, z_2)} \right| \geq \dot{\psi}_{\max};$$

- average acceleration is greater than the maximum allowed, i.e.,

$$\left| \frac{V_2 - V_1}{\mathrm{d}(z_1, z_2)} \right| \geq \dot{V}_{\max};$$

- z_2 is "behind" z_1, i.e., $(x_2, y_2) \in H_{(x_1,y_1),\psi_1}$, where $H_{(x_1,y_1),\psi_1}$ is the half-space with normal vector $(\cos(\psi_1), \sin(\psi_1))$ that has (x_1, y_1) on its boundary.

This estimate of the reachable set is a conservative estimate for all small T, i.e., for any $z \in X$, there exists some $T \in (0, \infty)$ such that $\mathcal{R}_t(z) \subseteq \hat{\mathcal{R}}_t(z)$ for all $t \in [0, T]$.

Branch-and-Bound

For `CostToGo`(z) in the branch-and-bound algorithm, the minimum distance from each node to the goal region is divided by the maximum achievable velocity so that the function never overestimates the actual cost.

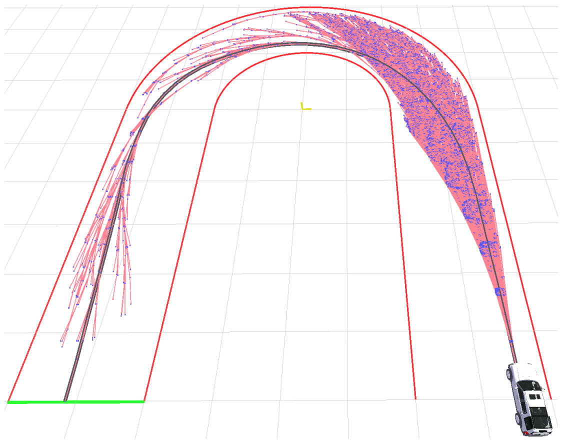
3.4 Simulation Results

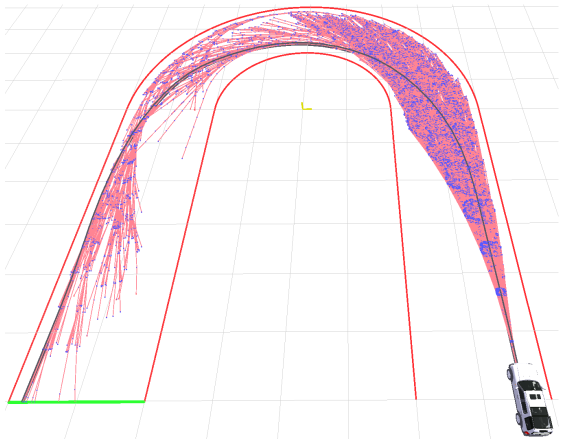
The algorithm is evaluated in 90-, 150-, 180-, and 270-degree turns, on a laptop with Intel® Core™2 Extreme Q9300 @ 2.53GHz processor and 4GB RAM. The time to complete the maneuver by reaching the end of the road is used as the cost function.

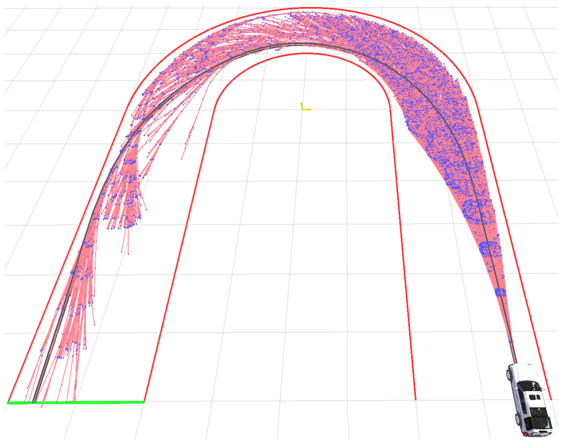
Figure 3-2 shows the RRT* tree (projected on the x-y coordinate space) for minimum-time 180-deg cornering. The solid black line represents the solution trajectory of the center of gravity. The branch-and-bound procedure makes the part of the tree close to the finish line relatively sparse due to pruning.

Figure 3-3 shows the anytime behavior of the RRT* algorithm applied to the cornering maneuver. The solution quality is improved as more computation is allowed.

Figure 3-4 compares trajectories, speeds, and vehicle slip angles of the turning maneuver for several turning angles. Plots show some regional non-smoothness because the algorithm did not grow the total number of vertices in the tree more than 60,000. It is noticeable that the time-optimal solution of 150-deg turning does not involve much skidding while solutions for other angles heavily involve the skidding regime. We observe that the characteristics of the optimal maneuver depends on the road shape and other conditions. Roughly speaking, trail-braking maneuvers by input parametrization in [115] can be characterized as two synchronized V-shapes in the speed and the slip angle. Our time-optimal 180-deg turning maneuver shows the synchronization of two V-shapes, meaning that the time-optimal maneuver for 180-deg turning (with an initial speed 60 km/h on a loose surface with a friction coefficient


(a) 11793 Nodes


(b) 20946 Nodes


(c) 30520 Nodes

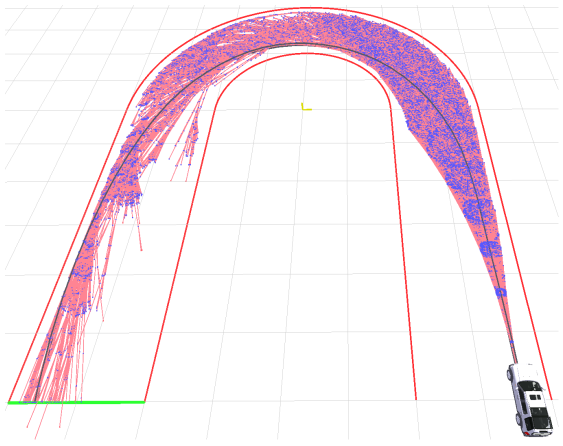

(d) 59542 Nodes

Figure 3-2: The RRT* Tree for 180-deg Turning

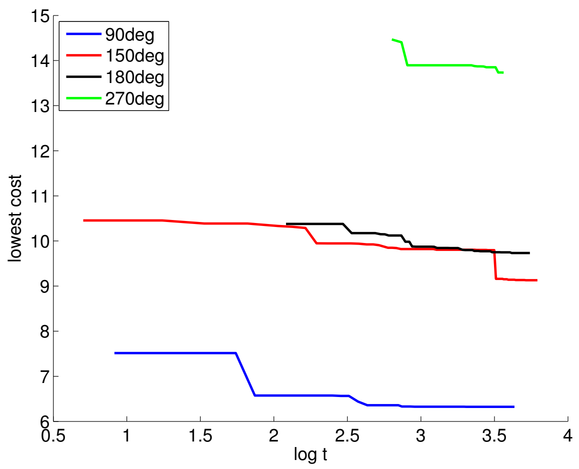
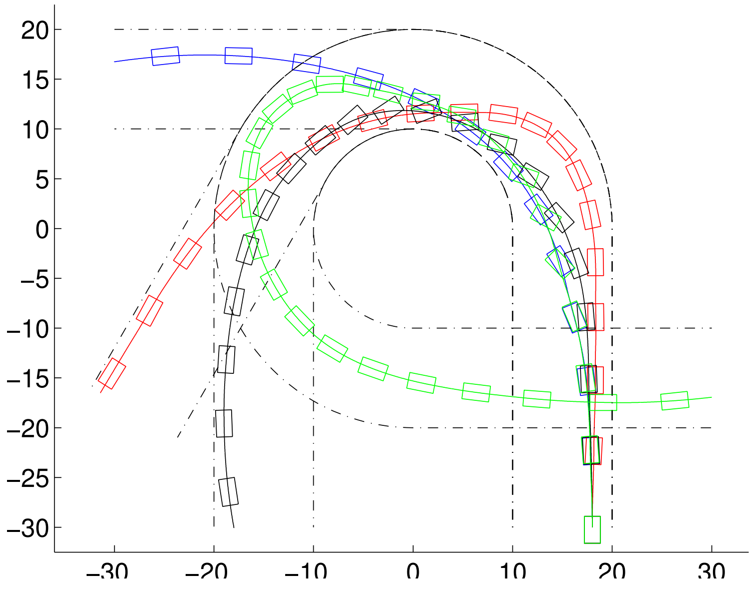
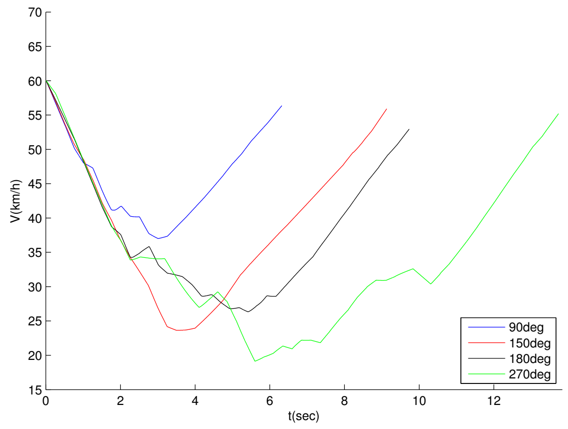


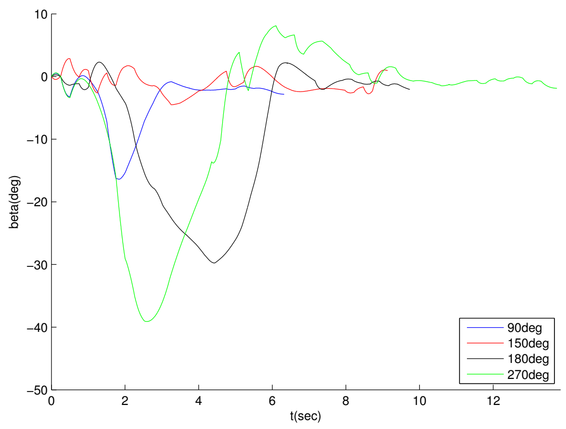
Figure 3-3: Anytime Computation for 90, 150, 180, and 270-deg Turning



(a) Trajectories. Vehicle is shown at every 0.5s.



(b) Vehicle Speed



(c) Vehicle Slip Angle

Figure 3-4: Comparison Plots for 90, 150, 180, and 270-deg Turning

$\mu = 0.52$) closely reproduces the trail-braking maneuver. For other road conditions and initial speeds, we expect other turning angles would result in the trail-braking maneuver as the time-optimal solution.

3.5 Conclusion

This chapter extended the application domain of the RRT* to systems with complex differential and geometric constraints with high-dimensional state spaces. The proposed methodology was applied to generate aggressive skidding maneuvers as minimum-time solutions for high-speed off-road vehicle cornering on loose surfaces with a low friction coefficient.

In this work, any rewired vertex with motion gaps immediately triggered the repropagation procedure (Algorithm 3.2). Instead, readers may choose to ignore gaps in the tree or delay the repropagation as in [76]. On the other hand, if more computational resources and efficient sub-routines become available after years, readers may choose to enhance the approximate TPBVP solver into an optimization-based TPBVP solver as in [118].

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 4

Semi-Analytic Local Steering for Pseudo-Flat Systems

In this chapter, we provide a methodology for a class of dynamical systems that are differentially flat under certain assumptions, i.e. "pseudo-flat" systems. As an example, we discuss an implementation of the RRT* for the half-car dynamical model similar to Chapter 3. To develop fast solutions of the associated local steering problem, we observe that the motion of a special point (namely, the front center of oscillation) can be modeled as a double integrator augmented with fictitious inputs. We first map the constraints on tire friction forces to constraints on these augmented inputs, which provides instantaneous, state-dependent bounds on the curvature of geometric paths feasibly traversable by the front center of oscillation. Next, we map the vehicle's actual inputs to the augmented inputs. The local steering problem for the half-car dynamical model can then be transformed to a simpler steering problem for the front center of oscillation, which we solve efficiently by first constructing a curvature-bounded geometric path and then imposing a suitable speed profile on this geometric path. Finally, we demonstrate the efficacy of the proposed motion planner via numerical simulation results.

4.1 Introduction

Motion planning for autonomous mobile vehicles [26] has traditionally focused on relatively simple unicycle-type kinematic models, owing both to the sufficiency of the resultant plans for low-speed vehicle motion, and to the computational efficiency afforded by these models. However, the resultant trajectories do not exploit the vehicle's maneuvering capabilities, and motion planning based on such vehicle models is unsuitable for enabling autonomous high-speed motion of car-like vehicles in complex and dynamic environments.

Motion planning for higher-dimensional, higher-fidelity vehicle dynamical models is, in general, difficult. Whereas computationally efficient motion planning in high-dimensional state spaces is made possible via randomized sampling-based algorithms [60, 74], these algorithms ignore the quality of the resultant motion, and often result in highly sub-optimal motion plans. Recent developments in optimal randomized sampling-based planning [57], and in deterministic approaches that include vehicle dynamical constraints [27] promise fast computation of near-optimal paths with higher-fidelity vehicle dynamical models. Both of these approaches, however, rely on the availability of a *local steering* algorithm – a two-point boundary value problem (TPBVP) solver – that computes a near-optimal control input to steer the vehicle between specified initial and final states, neither of which is necessarily an equilibrium state.

The main contributions of this chapter are a fast local steering algorithm specific to the half-car dynamical model [80], which captures yaw dynamics and normal load transfer of wheeled vehicles, and the implementation of the RRT* motion planning algorithm using this local steering algorithm.

4.1.1 Motivation and Related Work

Whereas the local steering problem is difficult to solve in general, in some specific cases, it is possible to exploit the structure of the dynamical system to develop a fast solution algorithm suitable for real-time implementations. In particular, the property of *differential flatness* [35] may be advantageously used. The states and inputs of differentially flat dynamical systems can be fully recovered from the so-called *flat outputs* of the system (and their derivatives).

In the context of motion planning, differential flatness of the vehicle dynamics model is particularly useful when the flat outputs are workspace[1] coordinates of a point associated with the vehicle. In such cases, the vehicle dynamical behavior may be inferred from workspace trajectories of this point. Conversely, the problem of motion planning subject to vehicle dynamical constraints may be transformed to a workspace trajectory generation problem, which is beneficial owing to the low dimensionality of the workspace and to the ease of collision checking with (workspace) obstacles.

For a half-car model of front-steered vehicles that incorporates wheel slip, special points associated with the model, called the *front and rear Huygens centers of oscillation* (CO) are associated with differential flatness [36, 38, 91]. A crucial physical property of a CO is that its acceleration is independent of one of the lateral tire friction forces [1]. The coordinates in a body-fixed frame of the velocity of the rear CO have been identified as flat outputs of a half-car model that incorporates wheel slip but does not consider normal load transfer [38].

More pertinent to motion planning, the coordinates in an inertial frame of the *position* of the front CO have been identified as "pseudo-flat" outputs for the half-car model [91]. The mapping from these outputs and their derivatives to the states and inputs of the vehicle involve not only algebraic equations, but also differential equations. The position coordinates of front and/or rear CO have been considered as reference inputs for trajectory tracking controllers in [102] (rear CO), in [47] (both front and rear CO with four-wheel steering), and in [63] (front CO). The notion of "pseudo-flatness" is closely related to the notion of *near-identity diffeomorphisms* [84], wherein stabilization and tracking of non-holonomic systems are discussed. The crucial difference between the proposed work and [84] is that we address the more involved TPBVP of optimal trajectory generation. In this context, numerical optimal control

[1]The *workspace* is the planar region in which the mobile vehicle operates.

techniques have been applied in [115] for reproducing a trail-braking maneuver, and in [23] for generating a minimum-time double lane-change maneuver. The generation of a minimum-time speed profile for a half-car traversing a given geometric path has been addressed in [112]. Preliminary results on the implementation of the RRT* algorithm for the half-car model have appeared in [51], where the local steering problem was solved numerically in a lower dimensional state space of the vehicle.

4.1.2 Contributions

The main contributions of this chapter are as follows. Firstly, we provide a fast local steering algorithm for the half-car dynamical model, which may be applied independently in conjunction with motion planners different from that considered in this chapter (RRT*). The key idea that enables this fast local steering is its separation into a geometric path planning step and an optimal time parametrization step, while always maintaining guarantees of feasibility vis-a-vis the dynamical constraints and input constraints of the half-car model. This separation provides a significant advantage in the implementation of the RRT*: rough collision checking can be performed after the geometric path planning step, and the computationally intensive optimal time-parametrization step can be skipped for cases where collisions are detected. The proposed local steering algorithm and the resulting RRT*-based motion planner constitute vast improvements in computation time over the implementation of [51], and enables the solutions of problems that were found to be impractically slow to solve using the approach of [51].

Secondly, we drop the simplifications to the half-car model adopted in [38, 91]: specifically, we allow *normal load transfer* between the front and rear tires – a phenomenon commonly utilized by rally racing drivers to control the yaw dynamics [115] – and we consider as inputs the longitudinal tire slips instead of the longitudinal tire forces. Manipulating the longitudinal tire slips (with thrust/brakes) is more realistic than manipulating longitudinal forces because these forces depend on the *total* tire slips, not the longitudinal slips alone.

Thirdly, we map the constraints on tire friction forces to equivalent constraints on the flat output trajectory. This mapping is itself a novelty in the context of differential flatness-based trajectory generation and control algorithms for the half-car, because friction force constraints are ignored in similar earlier works [38, 91].

Finally, we study an implementation of the RRT* with the proposed local steering algorithm, which is a fundamental way to develop an autonomous high-speed driving system that fully utilizes the vehicle's maneuvering capabilities.

The rest of this chapter is organized as follows. In Section 4.2, we describe the half-car dynamics model and discuss its differential flatness properties. In Section 4.3, we discuss an efficient local steering algorithm for the half-car model. In Section 4.4, we provide simulation results of the said implementation of the RRT*. Finally, we conclude the chapter in Section 4.5 with remarks.

4.2 The Half-Car Model

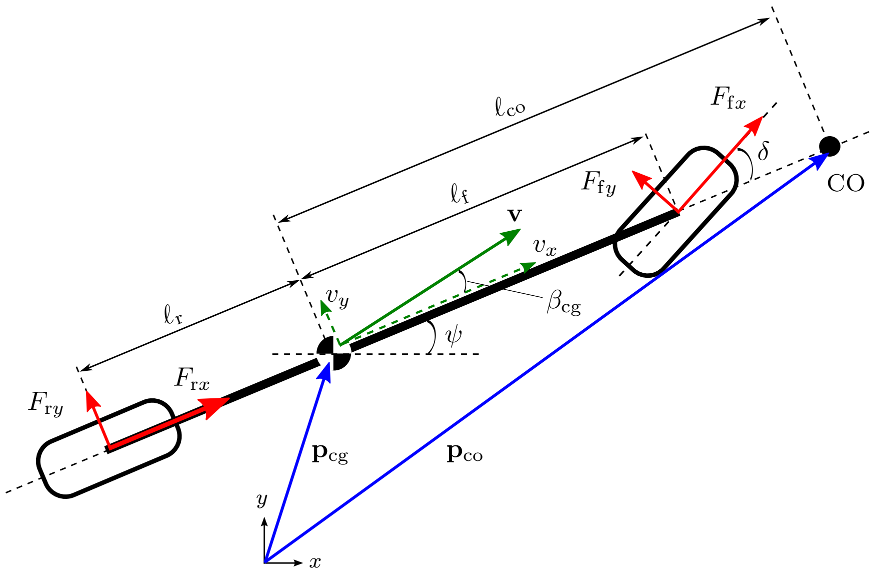


Figure 4-1: The half-car dynamical model: position vectors are in blue, velocity vectors are in green, and forces are in red color.

The half-car dynamical model is used in applications where the vehicle's position, heading, and sideslip are of primary interest (cf. [80, 38, 115, 91] and references therein). We rearrange the half-car model presented in Chapter 3, in a way that the equations lead to a simpler form. A half-car model is shown in Figure 4-1, with mass m, and yaw moment of inertia I_z. We denote by $\mathbf{p}_{\mathrm{cg}}$ the position vector of the center of gravity (CG) with respect to a pre-specified inertial axis system; by ψ the heading of the vehicle, and by v_{x} and v_{y} the components in a body-fixed axis system of the velocity $\mathbf{v}$ of the CG. We denote by ℓ_{f} and ℓ_{r}, respectively, the distances of the centers to the front and rear wheels from the center of gravity; by h the height of the CG; and by $F_{\alpha\beta}$, $\alpha \in \{\mathrm{f}, \mathrm{r}\}$, $\beta \in \{x, y\}$, the components in axes attached to the tires (with the x-axis in the plane of the tire) of frictional forces of the front and rear tires. The equations of motion for the half-car model are:

$$\begin{aligned}
m\dot{v}_x &= (F_{\mathrm{f}x}\cos\delta - F_{\mathrm{f}y}\sin\delta + F_{\mathrm{r}x}) + mv_y\dot{\psi}, && (4.1)\\
m\dot{v}_y &= (F_{\mathrm{f}x}\sin\delta + F_{\mathrm{f}y}\cos\delta + F_{\mathrm{r}y}) - mv_x\dot{\psi}, && (4.2)\\
I_z\ddot{\psi} &= \ell_{\mathrm{f}}\left(F_{\mathrm{f}x}\sin\delta + F_{\mathrm{f}y}\cos\delta\right) - \ell_{\mathrm{r}}F_{\mathrm{r}y}, && (4.3)
\end{aligned}$$

where δ is the steering angle of the front wheel, which we consider a control input. In what follows, we denote by ξ the state of the vehicle, i.e., $\xi = (p_{\mathrm{cg},x}, p_{\mathrm{cg},y}, \psi, v_x, v_y, \dot{\psi})$. The lateral slips $s_{\mathrm{f}y}$ and $s_{\mathrm{r}y}$ of the front and rear tires are:

$$s_{\mathrm{f}y} = \frac{(v_y + \ell_{\mathrm{f}}\dot{\psi})\cos\delta - v_x\sin\delta}{v_x\cos\delta + (v_y + \ell_{\mathrm{f}}\dot{\psi})\sin\delta}, \qquad (4.4)$$

$$s_{\mathrm{r}y} = \frac{v_y - \ell_{\mathrm{r}}\dot{\psi}}{v_x}, \qquad (4.5)$$

We consider as control inputs the longitudinal tire slips $s_{\mathrm{f}x}$ and $s_{\mathrm{r}x}$. The total tire slips are then given by $s_\alpha = \sqrt{s_{\alpha x}^2 + s_{\alpha y}^2}$, $\alpha \in \{\mathrm{f}, \mathrm{r}\}$, and the tire friction forces $F_{\alpha\beta}$ are:

$$F_{\alpha\beta} = \mu_{\alpha\beta}F_{\alpha z}, \qquad \alpha \in \{\mathrm{f}, \mathrm{r}\},\ \beta \in \{x, y\}, \qquad (4.6)$$

where $F_{\alpha z}$ are the normal tire loads given by (cf. [115]):

$$\begin{aligned}
F_{\mathrm{f}z} &= \frac{mg\left(\ell_{\mathrm{r}} - \mu_{\mathrm{r}x}h\right)}{\ell_{\mathrm{f}} + \ell_{\mathrm{r}} + h\left(\mu_{\mathrm{f}x}\cos\delta - \mu_{\mathrm{f}y}\sin\delta - \mu_{\mathrm{r}x}\right)}, && (4.7)\\
F_{\mathrm{r}z} &= mg - F_{\mathrm{f}z}, && (4.8)
\end{aligned}$$

and $\mu_{\alpha\beta}$ are coefficients given by Pacejka's formula [8]:

$$\mu_{\alpha\beta} := -\frac{s_{\alpha\beta}}{s_\alpha}\mu_\alpha \tag{4.9}$$

$$\text{with} \quad \mu_\alpha(s_\alpha) = D_\alpha \sin\left(C_\alpha \tan^{-1}(B_\alpha s_\alpha)\right), \tag{4.10}$$

for $\alpha \in \{\mathrm{f}, \mathrm{r}\}$, $\beta \in \{x, y\}$, where B_α, C_α, and D_α are constants. Note that (4.7)-(4.8) capture the load transfer effect, i.e., the normal tire loads depend upon the front and rear longitudinal tire slips, which relate to thrust/brake inputs.

Following the work of Peters *et al.* [91], we consider as a candidate flat output the position $\mathbf{p}_{\mathrm{co}}$ of the front CO [1], which is a point defined by

$$\mathbf{p}_{\mathrm{co}} = \begin{bmatrix} p_{\mathrm{co},x} \\ p_{\mathrm{co},y} \end{bmatrix} := \mathbf{p}_{\mathrm{cg}} + R(\psi)\begin{bmatrix} \ell_{\mathrm{co}} \\ 0 \end{bmatrix}, \tag{4.11}$$

where $\ell_{\mathrm{co}} := I_z/m\ell_{\mathrm{r}}$ and $R(\psi)$ is the rotation matrix. It may be shown (cf. [91]) that

$$\ddot{\mathbf{p}}_{\mathrm{co}} = R(\psi)\begin{bmatrix} \dot{v}_x - \dot{v}_y\dot{\psi} - \ell_{\mathrm{co}}\dot{\psi}^2 \\ \dot{v}_y + \dot{v}_x\dot{\psi} + \ell_{\mathrm{co}}\ddot{\psi} \end{bmatrix}. \tag{4.12}$$

We designate as an augmented input $\mathbf{u} := \ddot{\mathbf{p}}_{\mathrm{co}}$ the inertial acceleration $\ddot{\mathbf{p}}_{\mathrm{co}}$ of the CO, and we denote by $(u_{\mathrm{t}}, u_{\mathrm{n}})$ the body-axis coordinates of the augmented input. To map the augmented input to the vehicle control inputs $(s_{\mathrm{f}x}, s_{\mathrm{r}x}, \delta)$, note that, by (4.1)-(4.3) and (4.12),

$$\begin{bmatrix} u_{\mathrm{t}} \\ u_{\mathrm{n}} \end{bmatrix} = \frac{1}{m}\begin{bmatrix} F_{\mathrm{f}x}\cos\delta - F_{\mathrm{f}y}\sin\delta + F_{\mathrm{r}x} - m\ell_{\mathrm{co}}\dot{\psi}^2 \\ (\ell_{\mathrm{f}} + \ell_{\mathrm{r}})\left(F_{\mathrm{f}x}\sin\delta + F_{\mathrm{f}y}\cos\delta\right)/\ell_{\mathrm{r}} \end{bmatrix}. \tag{4.13}$$

Firstly, observe that (4.13) is an under-determined system of equations, which implies that the three vehicle control inputs $(s_{\mathrm{f}x}, s_{\mathrm{r}x}, \delta)$ cannot be determined uniquely from the trajectory $t \mapsto \mathbf{p}_{\mathrm{co}}(t)$ of CO. (Note that the forces $F_{\alpha\beta}$, $\alpha \in \{\mathrm{f}, \mathrm{r}\}, \beta \in \{x, y\}$ depend on $s_{\mathrm{f}x} and s_{\mathrm{r}x}$ via (4.4)-(4.10).) However, we may treat one of the three inputs as an "exogenous" input, and subsequently determine the other two inputs.

Secondly, observe that the computation of ψ and $\dot{\psi}$ involves the solution of the ODE (4.3). Consequently, the mapping from $\mathbf{p}_{\mathrm{co}}$ and its derivatives to the vehicle control inputs is a system of coupled algebraic-differential equations, and $\mathbf{p}_{\mathrm{co}}$ may thus be considered a "pseudo-flat" output. In what follows, we outline an analytic solution, if one exists, of (4.13), without recourse to numerical means of solution.

To this end, we first non-dimensionalize the physical quantities involved by dividing all lengths by $\ell_{\mathrm{f}} + \ell_{\mathrm{r}}$, all velocities by $\sqrt{g(\ell_{\mathrm{f}} + \ell_{\mathrm{r}})}$, all angular velocities by $\sqrt{g/(\ell_{\mathrm{f}} + \ell_{\mathrm{r}})}$, all accelerations by g, and all forces by mg. In a minor abuse of notation, all symbols in the remainder of the chapter represent non-dimensionalized quantities. For analytical simplicity, we assume as an exogenous input the rear tire longitudinal slip $s_{\mathrm{r}x}$, which may be manipulated independently of $\mathbf{p}_{\mathrm{co}}(t)$. To compute the other two control inputs – the steering angle δ and the front tire longitudinal slip $s_{\mathrm{f}x}$ – after selecting $s_{\mathrm{r}x}$, we perform the following computations.

Equations (4.6)–(4.9) may be manipulated to show that

$$F_{\mathrm{f}z} = -\frac{h}{\ell_{\mathrm{f}} + \ell_{\mathrm{r}}} \left(u_{\mathrm{t}} - \frac{\ell_{\mathrm{r}}}{h} + \ell_{\mathrm{co}} \dot{\psi}^2 \right), \tag{4.14}$$

and that

$$R(\delta) \begin{bmatrix} s_{\mathrm{f}x} \\ s_{\mathrm{f}y} \end{bmatrix} \frac{\mu_{\mathrm{f}}}{s_{\mathrm{f}}} = - \begin{bmatrix} \sigma_1(\xi, s_{\mathrm{r}x}) \\ \sigma_2(\xi) \end{bmatrix}, \tag{4.15}$$

where the maps $\sigma_1(\xi, s_{\mathrm{r}x})$ and $\sigma_2(\xi)$ are defined as follows:

$$\sigma_1(\xi, s_{\mathrm{r}x}) := \frac{1}{h} \left(\frac{\ell_{\mathrm{r}} - h\mu_{\mathrm{r}x}}{F_{\mathrm{f}z}} - (\ell_{\mathrm{f}} + \ell_{\mathrm{r}}) \right) + \mu_{\mathrm{r}x}, \tag{4.16}$$

$$\sigma_2(\xi) := \frac{\ell_{\mathrm{r}} u_{\mathrm{n}}}{(\ell_{\mathrm{f}} + \ell_{\mathrm{r}}) F_{\mathrm{f}z}}. \tag{4.17}$$

We may compute the friction coefficient μ_{f} of the front tire by (4.15) as

$$\mu_{\mathrm{f}} = \sqrt{\sigma_1^2(\xi, s_{\mathrm{r}x}) + \sigma_2^2(\xi)}, \tag{4.18}$$

and the total slip s_{f} of the front tire may then be computed from (4.10). After further algebraic manipulations, we arrive at the following equation in δ:

$$\begin{aligned}- v_x s_{\mathrm{f}} \sigma_2(\xi) \cos^2 \delta + (v_y + \ell_{\mathrm{f}} \dot{\psi}) s_{\mathrm{f}} \sigma_1(\xi, s_{\mathrm{r}x}) \sin^2 \delta \\ + (v_x \sigma_1(\xi, s_{\mathrm{r}x}) - (v_y + \ell_{\mathrm{f}} \dot{\psi}) \sigma_2(\xi)) s_{\mathrm{f}} \sin \delta \cos \delta \\ - (v_y + \ell_{\mathrm{f}} \dot{\psi}) \mu_{\mathrm{f}} \cos \delta + v_x \mu_{\mathrm{f}} \sin \delta = 0. \quad (4.19)\end{aligned}$$

Following an appropriate transformation of variables, (4.19) may be transformed to a quartic polynomial equation, which may be solved analytically for δ. Finally, the longitudinal slip $s_{\mathrm{f}x}$ of the front tire may be computed using (4.4).

4.3 Local Steering for the Half-Car Model

Informally, the problem of optimal motion planning involves the determination of admissible control inputs for a nonlinear dynamical system such that (a) the state of the system is transferred from a pre-specified initial state ξ_0 to a pre-specified final state ξ_{f}, (b) the resultant state trajectory does not intersect with a pre-specified subset of the state space, called the *obstacle space*, and (c) a pre-specified integral cost is minimized along the resultant state trajectory.

The RRT* algorithm [57] solves the optimal motion planning problem by constructing a tree of state trajectories of the system. Each vertex of this tree is associated with a state of the nonlinear dynamical system, and each edge is associated with an admissible control input. Initially the aforesaid tree contains only one vertex associated with the initial state ξ_0. At each subsequent iteration, the RRT* algorithm samples a new state, extends the tree towards this state, and attempts to reassign the parent of each nearby vertex. The edge construction between vertices is achieved by a *local steering algorithm* (which we denote by STEER).

We note the following key issues [57]: (a) the number of times that the RRT* invokes STEER is $\mathcal{O}(\log n)$ per iteration, or $\mathcal{O}(n \log n)$ overall; and (b) the state trajectory that corresponds to the control input found by STEER is subject to a further collision check to be included in the collision-free tree. It follows that a fast STEER is crucial to the speed of the overall motion planner; furthermore, the computation time expended for instances of STEER that fail the subsequent collision check may be counted as "wasted" time, because these execution instances do not further advance the motion planner.

To design a fast STEER, we leverage the "pseudo-flat" nature of the system to transform the steering problem for the half-car model to a steering problem for the simpler particle model $\ddot{\mathbf{p}}_{\rm co} = \mathbf{u}$ that describes the motion of the CO. To this end, we map the constraints on the tire friction forces to equivalent constraints on the augmented input u_x. These constraints on u_x impose bounds on the lateral acceleration of the CO, which in turn correspond to speed-dependent curvature bounds on paths that the CO can feasibly traverse. Next, we approximate a time-optimal trajectory for the CO by first constructing a curvature-bounded geometric path and then by imposing a minimum-time speed profile on this path. Finally, we determine the acceleration u_x of the CO for tracking this trajectory, and we map u_x to the vehicle inputs using the computational procedure outlined in Section 4.2.

As we will discuss in Section 4.4, the proposed approach for STEER is faster than a numerical optimal control-based approach. Furthermore, it also enables significant savings of the aforementioned "wasted" computation time by allowing collision checks to be performed after the (fast) geometric path planning step and before the (relatively slow) optimal time parametrization step.

4.3.1 Constraints on Pseudo-Flat Output Trajectories

The magnitude F_α, $\alpha \in \{\mathrm{f}, \mathrm{r}\}$, of the total friction force at each tire, depends on the friction coefficient and the normal load on that tire: $F_\alpha = \mu_\alpha F_{\alpha z}$. It follows that $F_\alpha = \sqrt{F_{\alpha x}^2 + F_{\alpha y}^2} \leq \mu_\alpha^* F_{\alpha z}$, where μ_α^* is the maximum value of the tire friction coefficient. In what follows, we show that the acceleration u_x of the CO is constrained to lie within an ellipse, the dimensions of which depend on the vehicle state (in particular, the sideslip $\beta_{\rm cg} := \tan^{-1}(v_y/v_x)$, and the yaw rate $\dot{\psi}$), the maximum value $\mu_{\rm f}^*$ of the front tire friction coefficient, and the rear tire longitudinal slip $s_{{\rm r}x}$, which was chosen in Section 4.2 as an "exogenous" input.

To this end, let $k_1 := -h/(\ell_{\mathrm{f}} + \ell_{\mathrm{r}})$, $k_2 := (\ell_{\mathrm{f}} + \ell_{\mathrm{r}})/\ell_{\mathrm{r}}$, and define the map

$$\sigma_3(\xi) := k_1(\ell_{\mathrm{co}}\dot{\psi}^2 - \ell_{\mathrm{r}}/h). \tag{4.20}$$

Note that, by (4.14),

$$F_{\mathrm{f}z} = k_1 u_{\mathrm{t}} + \sigma_3(\xi), \tag{4.21}$$

and that, by (4.13),

$$u_{\mathrm{t}}^2 + \frac{u_{\mathrm{n}}^2}{k_2^2} = (F_{\mathrm{f}x}\cos\delta - F_{\mathrm{f}y}\sin\delta + F_{\mathrm{r}x} - \ell_{\mathrm{co}}\dot{\psi}^2)^2 + (F_{\mathrm{f}x}\sin\delta + F_{\mathrm{f}y}\cos\delta)^2. \tag{4.22}$$

The R.H.S. of (4.22) involves the front and rear tire lateral and longitudinal forces, and we may use (4.6), (4.9), and (4.21) to express the R.H.S. of (4.22) in terms of ξ, $s_{\mathrm{r}x}$, and μ_{f} to arrive at the following equation:

$$\left(\frac{u_{\mathrm{t}} + \sigma_4(\xi, s_{\mathrm{r}x}, \mu_{\mathrm{f}})}{\sigma_5(\xi, s_{\mathrm{r}x}, \mu_{\mathrm{f}})}\right)^2 + \left(\frac{u_{\mathrm{n}}}{\sigma_6(\xi, s_{\mathrm{r}x}, \mu_{\mathrm{f}})}\right)^2 = 1, \tag{4.23}$$

where σ_4, σ_5, and σ_6 are maps whose detailed expressions are provided as

$$\sigma_4(\xi, s_{\mathrm{r}x}, \mu_{\mathrm{f}}) := \frac{(\mu_{\mathrm{r}x}k_1 + 1)(\ell_{\mathrm{co}}\dot{\psi}^2 + \mu_{\mathrm{r}x}(\sigma_3 - 1)) - \mu_{\mathrm{f}}^2 k_1 \sigma_3}{\sigma_7^2}, \tag{4.24}$$

$$\sigma_5(\xi, s_{\mathrm{r}x}, \mu_{\mathrm{f}}) := \frac{\sigma_6}{k_2\sigma_7}, \tag{4.25}$$

$$\sigma_6(\xi, s_{\mathrm{r}x}, \mu_{\mathrm{f}}) := k_2\sqrt{\mu_{\mathrm{f}}^2\sigma_3^2 - (\ell_{\mathrm{co}}\dot{\psi}^2 + \mu_{\mathrm{r}x}(\sigma_3 - 1))^2 + (\sigma_4\sigma_7)^2}, \tag{4.26}$$

$$\sigma_7(\xi, s_{\mathrm{r}x}, \mu_{\mathrm{f}}) := \sqrt{(\mu_{\mathrm{r}x}k_1 + 1)^2 - \mu_{\mathrm{f}}^2 k_1^2}. \tag{4.27}$$

Note that the values of these maps define the location of the center and the dimensions of an ellipse in the $u_{\mathrm{t}} - u_{\mathrm{n}}$ plane. The constraints on the individual tire friction forces may now be mapped to the following elliptical constraint[2] on the acceleration of the CO:

$$\left(\frac{u_{\mathrm{t}} + \sigma_4(\xi, s_{\mathrm{r}x}, \mu_{\mathrm{f}}^*)}{\sigma_5(\xi, s_{\mathrm{r}x}, \mu_{\mathrm{f}}^*)}\right)^2 + \left(\frac{u_{\mathrm{n}}}{\sigma_6(\xi, s_{\mathrm{r}x}, \mu_{\mathrm{f}}^*)}\right)^2 \leq 1. \tag{4.28}$$

[2]It is straightforward to show that for any $\mu_{\mathrm{f}_1}, \mu_{\mathrm{f}_2}$ with $\mu_{\mathrm{f}_1} \leq \mu_{\mathrm{f}_2}$, the ellipse defined by (4.23) with $\mu_{\mathrm{f}} = \mu_{\mathrm{f}_1}$ is completely contained within the ellipse defined by (4.23) with $\mu_{\mathrm{f}} = \mu_{\mathrm{f}_2}$.

Proposition 4.1 (Existence of real roots to (4.19)**)** *There exists at least one real root to* (4.19) *whenever* u_{t} *and* u_{n} *satisfy* (4.28)*.*

Proof The algebraic manipulations involved in arriving at (4.19) from (4.15)-(4.18) include the equation

$$s_{\mathrm{f}x}^2 + s_{\mathrm{f}y}^2(\xi, \delta) = \frac{1}{B_{\mathrm{f}}^2} \tan^2\left(\frac{1}{C_{\mathrm{f}}} \sin^{-1}\left(\frac{\mu_{\mathrm{f}}}{D_{\mathrm{f}}}\right)\right), \tag{4.29}$$

and it follows that

$$(v_y + \ell_{\mathrm{f}}\dot{\psi} \mp \sigma_8 v_x)\cos\delta + (\mp\sigma_8(v_y + \ell_{\mathrm{f}}\dot{\psi}) - v_x)\sin\delta = 0, \tag{4.30}$$

where

$$\sigma_8 := \sqrt{\frac{1}{B_{\mathrm{f}}^2} \tan^2\left(\frac{1}{C_{\mathrm{f}}} \sin^{-1}\left(\frac{\mu_{\mathrm{f}}}{D_{\mathrm{f}}}\right)\right) - s_{\mathrm{f}x}^2}. \tag{4.31}$$

Equation (4.19) is obtained from (4.30) by eliminating $s_{\mathrm{f}x}$ from (4.30) using (4.15), and consequently, the existence of real roots of (4.19) is equivalent to the existence of real roots of (4.30). It easy to show that, following an appropriate transformation of variables, that (4.30) is equivalent to a quadratic equation that has real roots whenever

$$(v_y + \ell_{\mathrm{f}}\dot{\psi} \mp \sigma_8 v_x)^2 + (\mp\sigma_8(v_y + \ell_{\mathrm{f}}\dot{\psi}) - v_x)^2 > 0, \tag{4.32}$$

which holds true whenever σ_8 is real, which in turn holds true whenever $|\mu_{\mathrm{f}}| \leq D_{\mathrm{f}} = \mu_{\mathrm{f}}^*$. By (4.18), and by the arguments leading to the constraint (4.28), it follows that $0 \leq \mu_{\mathrm{f}} \leq \mu_{\mathrm{f}}^*$ whenever u_{t} and u_{n} satisfy (4.28), and the result follows. $\square$

Informally, Proposition 4.1 states that whenever the commanded acceleration of the front CO satisfies the constraints imposed by the ground-tire friction force characteristics, there exists at least one solution to the system of nonlinear equations in (4.13). We reiterate that all computations involved in computing the vehicle control inputs $(s_{\mathrm{f}x}, s_{\mathrm{r}x}, \delta)$ from the acceleration $\mathbf{u}$ of the CO (i.e., the second derivatives of the flat output) are simple, and that none of these computations involve any computationally expensive numerical optimization or root-finding.

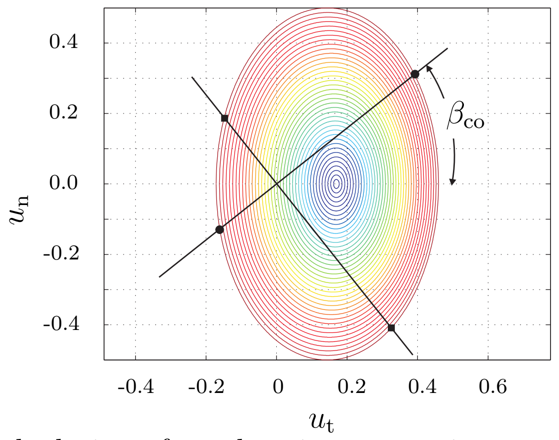


Figure 4-2: Calculation of acceleration constraints on the CO from (4.28).

Let β_co denote the sideslip of the CO, i.e., the angle between $\dot{\mathbf{p}}_\mathrm{co}$ and the body x-axis. It can be shown that $\beta_\mathrm{co} = \tan^{-1}\Big((v_y + \ell_\mathrm{co}\dot{\psi})/v_x\Big)$. Then, the intersections of the line of inclination β_co passing through the origin of the $u_\mathrm{t} - u_\mathrm{n}$ plane with the boundary of the elliptical region described by (4.28) provide the upper and lower bounds on the pure tangential acceleration of the CO. Similarly, the intersections of the line of inclination $\beta_\mathrm{co} + \frac{\pi}{2}$ passing through the origin of the $u_\mathrm{t} - u_\mathrm{n}$ plane with the boundary of the elliptical region described by (4.28) provide bounds on the pure lateral acceleration of the CO (see Figure 4-2).

Geometric paths traversed by a particle with bounded lateral acceleration have bounded curvature. Specifically, the curvature bounds for left and right turns, respectively, are $v^2/u_\perp^\mathrm{max}$ and $v^2/|u_\perp^\mathrm{min}|$, assuming $u_\perp^\mathrm{max} > 0$ and $u_\perp^\mathrm{min} < 0$. The construction of the shortest curvature-bounded path with asymmetric (about the origin) bounds on the curvature is straightforward: it has been shown in [9] that the shortest path is contained within the Dubins' family of paths. The Dubins' family of paths consists of continuously differentiable paths that are obtained by concatenating at most three sub-paths, each of which is either a straight line segment or a circular arc of radius of equal to the minimum radius of turn.

Similarly, the minimum-time speed profile on a prescribed curve for a particle with an elliptical acceleration constraint has been discussed in, for instance, [113]. Briefly, the approach in [113] involves determining switching points along the prescribed curve, such that, between two consecutive switches, the particle either travels with maximum possible tangential acceleration, or travels with maximum possible tangential deceleration, or travels at the critical speed. This critical speed is defined, point-wise along the prescribed curve, as the speed at which the centripetal acceleration required for the particle to change its direction of travel at the rate prescribed by the instantaneous curvature equals the maximum lateral acceleration of the particle.

4.4 Simulation Results and Discussion

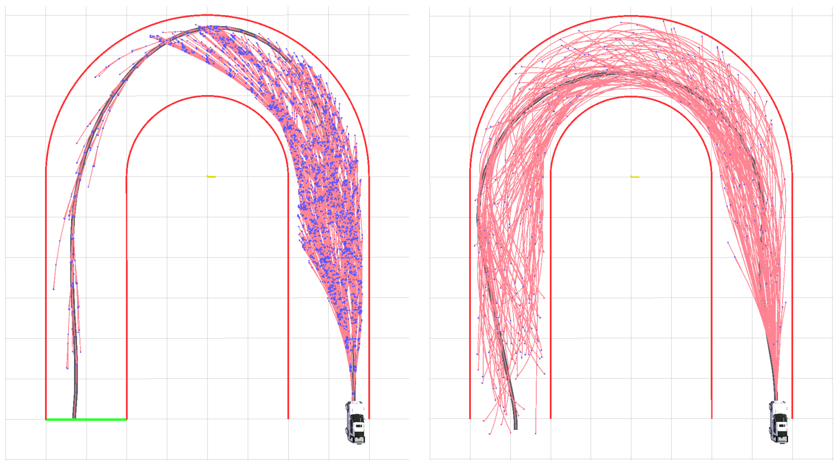
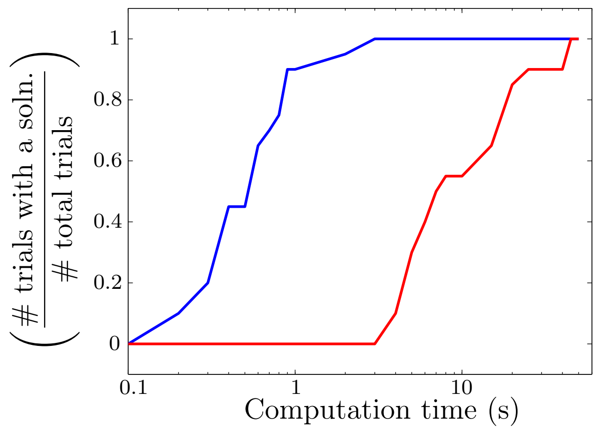

(a) From Chapter 3 (b) Proposed

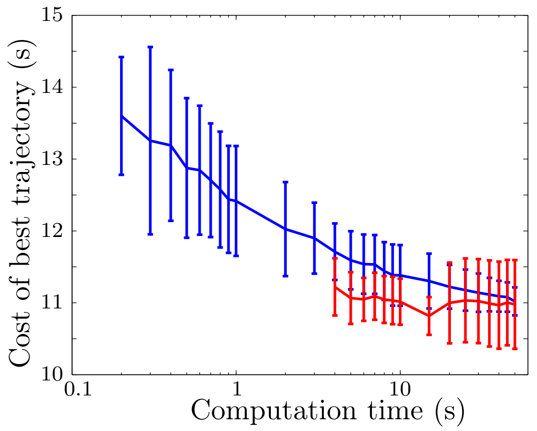
Figure 4-3: Trajectories obtained within a specified computation time when the RRT* uses two different implementations of Steer for the half-car dynamical model.

The proposed motion planner is fast, and its speed of execution makes it suitable for real-time implementations. To corroborate this claim, we present sample numerical simulation results for the proposed motion planner, including simulations with hard upper bounds on the execution time.

A preliminary implementation of the RRT* for the half-car dynamical model was discussed in Chapter 3, where numerical methods were used in a reduced dimensional state space to implement Steer. Figure 4-3 illustrates a sample simulation result comparing the coverage of the state space achieved by the RRT* algorithm implemented using the local steering method of Chapter 3 against that using the



(a) Ratio of finding at least one solution over 20 trials within time.



(b) Mean and standard deviation over 20 trials of the best known cost.

Figure 4-4: Execution speed and resultant trajectory costs of the RRT* motion planner with the proposed Steer (blue) compared to the Steer in Chapter 3 (red).

proposed method. Both of these algorithms were executed for a fixed period of time. As expected, the proposed implementation of the RRT* achieved significantly better coverage than that discussed in Chapter 3.

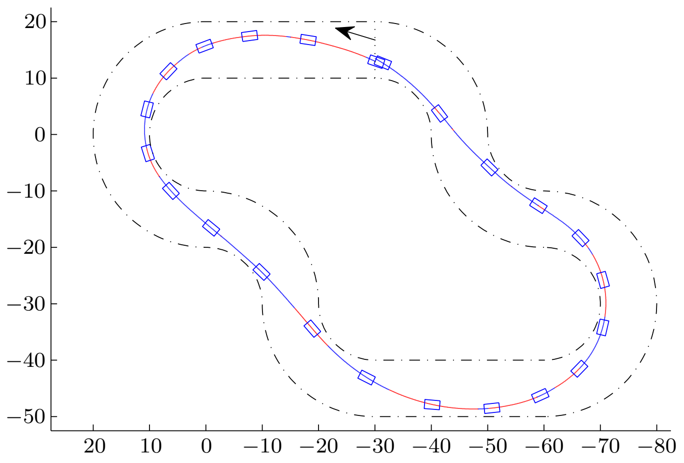
The data in Figure 4-4 was obtained over 20 trials for the problem of planning the 180-degree turn in Figure 4-3. In particular, Figure 4-4a shows that, within an execution time of 1s, no feasible trajectory was found in any of the trials by the implementation of Chapter 3, whereas feasible trajectories were found in all but two trials of the proposed implementation. In these simulations, the ratio of the average time required to find a first feasible solution with the implementation of Chapter 3 to that with the proposed implementation was 21.23. Moreover, the ratio of the *maximum* time required to find a first feasible solution with the implementation of Chapter 3 to the *minimum* time required with the proposed implementation was 226.3. Figure 4-4b shows the statistics for the costs of best trajectories achieved by the two implementations within specified execution times.

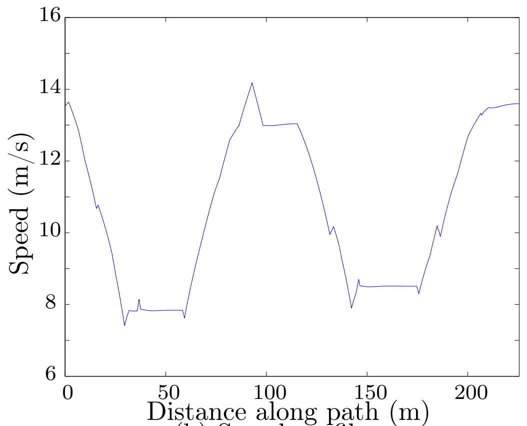
As discussed in Section 4.2, we chose the rear tire longitudinal slip $s_{\mathrm{r}x}$ as an "exogenous" input, and we set it to a constant value $s_{\mathrm{r}x,0}$. Consequently, the proposed implementation of the RRT* converges asymptotically to an optimal control input within the class of admissible control inputs with $s_{\mathrm{r}x} = s_{\mathrm{r}x,0}$. In comparison, the implementation of the RRT* in Chapter 3 converges asymptotically to a globally optimal control input (at the cost of slower execution).

The speed of execution of the proposed motion planner enabled the solution of problems that were found to be impractically slow with the approach in Chapter 3. For example, Figure 4-5 illustrates the application of the proposed approach to motion planning on a closed circuit, similar to a race track. Figure 4-5a illustrates the geometric path corresponding to a sample resultant trajectory, along with the vehicle's orientation (to indicate sideslip). Figure 4-5b shows the speed profile over the sample resultant trajectory, and Figure 4-5c shows the decreases in resultant trajectory cost with the progress of the algorithm, for three different trials. In addition, a speed profile planned for a more involved example is shown in Figure 4-6, on a closed circuit similar, but not identical, to the Monza F1 circuit [92].

As previously mentioned, the proposed STEER allows for a primary collision check (for the position of the front CO) to be performed immediately after the geometric path planning step. A secondary collision check (considering the finite size of the half-car and the heading angle) can be performed after the time-parametrization step. The computational advantage of this two-step collision check is that a large number of potential collisions can be detected *before* the relatively slow time-parametrization computations. For the particular case of 180-degree turn illustrated in Figure 4-3 over 20 trials, $98.76 \pm 0.05\%$ of the detected collisions were found immediately after the geometric path planning step. For the closed circuit case in Figure 4-5, the ratio was higher as $99.02 \pm 0.08\%$.

To anticipate future real-time implementations with hard bounds on the execution time, we implemented the solution of the closed circuit motion planning problem using a receding-horizon approach. In this approach, the motion planner first computes, within a pre-specified computation time t_{comp}, a trajectory over a pre-specified horizon of length along the circuit. Next, the vehicle's motion is simulated for a pre-specified execution time and the process is repeated. Figure 4-7 shows the total trajectory cost obtained by the aforesaid receding-horizon planner, over a range of values of t_{comp}. The total trajectory costs thus obtained are comparable to the trajectory costs


(a) Geometric path and vehicle orientation: the red segments indicate braking; the blue segments indicate acceleration.


(b) Speed profile

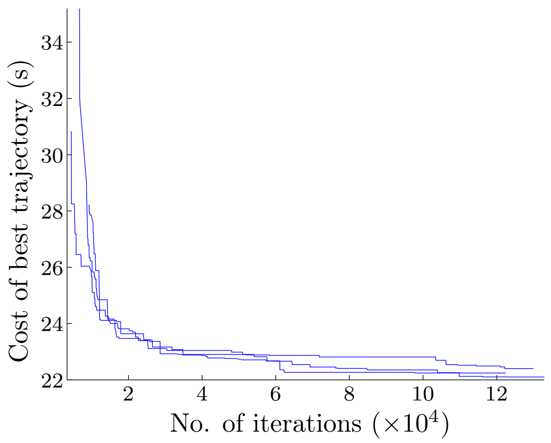

(c) Reductions in trajectory cost

Figure 4-5: Motion planning with the half-car model over a closed circuit.

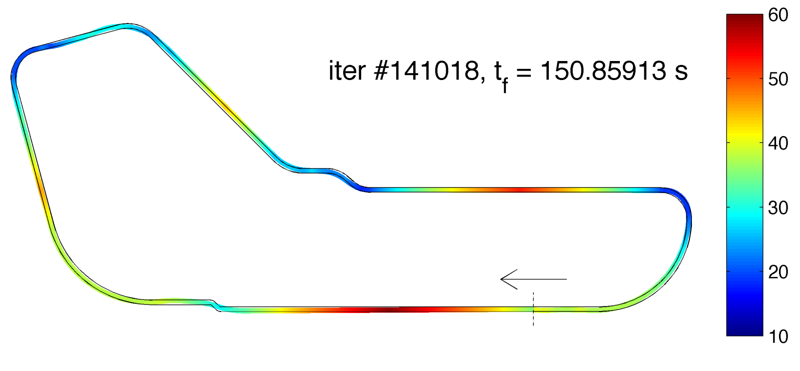


Figure 4-6: Speed profile in m/s on a closed circuit similar to the Monza track

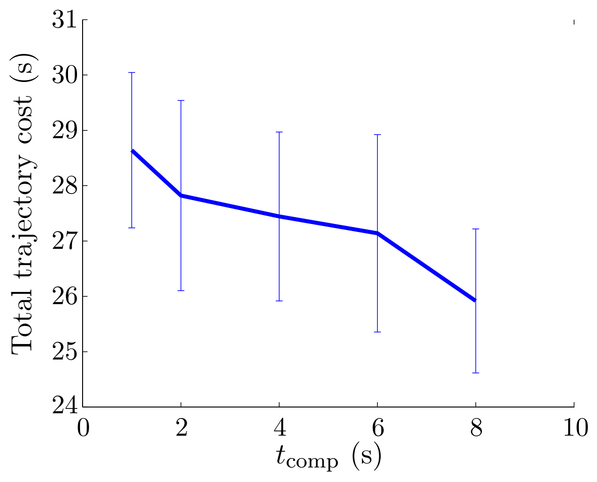


Figure 4-7: Total trajectory costs using a receding-horizon approach to motion planning over the closed circuit shown in Figure 4-5a.

shown in Figure 4-5. A similar planner using the implementation of Chapter 3 was unable to find feasible trajectories for any of these values of t_{comp}. The value t_{comp} implies t_{comp} seconds of computation on Intel® Core™2 Extreme Q9300 @ 2.53GHz processor with 4GB RAM.

4.5 Conclusions

In this chapter, we discussed a fast motion planner that incorporates the half-car dynamical model for wheeled vehicles. The key to an efficient implementation of the RRT* for the half-car model is a fast local steering algorithm that we introduced here. The constituent algorithms involved in the proposed local steering method—namely, the computation of a curvature-bounded geometric path, the imposition of a minimum-time speed profile, and the mapping of $\mathbf{u}$ to $(s_{\text{f}x}, s_{\text{r}x}, \delta)$—are all fast. Crucially, the proposed method for local steering, *by construction*, results in trajectories that are dynamically feasible and satisfy input constraints. Finally, the proposed approach enables motion planning in the space of the coordinates of $\mathbf{p}_{\text{co}}$ and $\dot{\mathbf{p}}_{\text{co}}$, instead of the full state space.

Chapter 5

Efficient Local Steering for Controllable Linear Systems

In this chapter, we propose an efficient method for local steering, based on polynomial basis functions and segmentation, with its application to controllable linear systems subject to linear state/input constraints. The main computational procedures for our flatness-based local steering include linear or quadratic programs with a small number of variables, where inequality constraints are added progressively after being identified by root-finding in low or medium order of polynomials.

5.1 Introduction

The RRT* [57], an asymptotically optimal extension of RRTs, provided remarkable answers to minimum-time and minimum-length motion planning problems of single integrators, double integrators, and Dubins' paths, by importing analytic solutions for optimal local steering problems from optimal control theory [21, 56]. Unfortunately, a *local steering* problem, or equivalently a *two-point boundary value problem (TPBVP)*, that steers a system between two specified states in a configuration space, does not generally allow analytic optimal solutions for dynamical systems with differential constraints. Recently, numerical [51] and semi-analytic [89, 117, 41, 43, 49] local steering methods with the RRT* have expanded the application areas of optimal motion planning. However, some of the following issues have been unavoidable in the absence of analytic TPBVP solutions: accumulated errors (or motion gaps) at tree vertices, implicitness of state/input trajectories, sub-optimal handling of state/input constraints, lack of generality in cost functionals, or impractically large computation. To avoid such issues, local steering methods need to be fast, exact at start/end states, and optimal, while strictly satisfying the imposed state/input constraints.

5.1.1 Contributions

In this chapter, we propose a flatness-based local steering algorithm for controllable linear systems with the following advantages: fast in computation, exact at both boundary states, and strictly constrained by linear state and input constraints. Opposed to optimization-centric approaches [111, 79, 32, 78] for differentially flat systems [35], we avoid a large optimization problem to better account for the non-convexity of obstacle-free space. We progressively add constraints by detecting locally-maximal violation of constraints, instead of adding constraints at many collocation points. Based on symbolic mappings into free coefficients of polynomial basis functions, linear or quadratic objective functions for the optimization over the free coefficients are determined by the cost functionals that combine terminal time and quadratic state/input penalties. Our mappings among states, inputs, and trajectories are consistent and exact without numerical integration, thus the obtained solutions are not subject to accumulated errors or motion gaps at both boundary states for which we solve the local steering problem. If the proposed algorithm solves TPBVPs in the RRT* and GR-FMTs, the obtained solution attains the asymptotic optimality within a set we define in Section 5.2.

5.1.2 Related Work

The constrained Linear Quadratic Regulator (LQR) [11], with its mathematical origin on Model Predictive Control (MPC) [22], possesses most of the capability that is needed to solve optimal motion planning problems, except obstacle avoidance.

Although MPC and its quadratic programming formulation have been remarkably practical and widely applicable, the non-convexity of obstacle-free space imposes fundamental difficulties to solution approaches that rely on a large optimization. On the contrary, approaches in [117, 41, 43] natively deal with the non-convex free space within the sampling-based framework, and the associated TPBVPs are solved semi-analytically by referring to optimal control theory. However, apart from the sensitivity to numerical issues, local steering methods that check the satisfaction of constraints afterwards and reject constraint-violating trajectories have difficulties in converging to the optimum given binding constraints.

Conceptually, progressive constraints in [33] play similar roles to our progressive procedure. A main distinction exists in that our application to linear systems and constraints is guaranteed to converge or terminate. Polynomial-based approaches in the past [95, 98, 86] share parts of formulations and techniques with our approach, but those approaches focus on generating a single extended trajectory fast.

LQR-trees [108] probabilistically cover the state space of interest, with a small number of optimization-based trajectories that are locally stabilizable by time-varying LQR controllers. The approach demonstrates provably-safe maneuver executions close to intended motions, whereas our work intends to keep obtaining new optimum from the current state, with potentially more computation and book-keeping. By construction, both approaches natively satisfy state and input constraints.

Markov Decision Processes (MDPs) [94] handle stochastic control scenarios and return feedback policies, but with dependence on discretization, approximation, or interpolation of time, state space, input space, or value function. Moreover, with discount factor $\gamma < 1$, MDPs minimize the cost functional with less weighting on future costs, thus the obtained solutions are different from the intended ones from the definition of the cost functional. As process noises vanish, maintaining the same level of consistency among time, state, input, and trajectory requires more computation or storage due to discretization issues.

This chapter is organized as follows. Section 5.2 provides supplementary definitions, in addition to definitions in Chapter 1. Section 5.3 proposes a local steering algorithm for controllable linear systems subject to linear state/input constraints. Section 5.4 shows simulation results with proposed algorithms, and we conclude with remarks in Section 5.5.

5.2 Definitions

We closely follow our previous definitions in Chapter 1, with some supplementary definitions. In this chapter, we elaborate the cost functional (1.3)

$$J(\mathbf{x}, \mathbf{u}) = h(x(T)) + \int_0^T g(x(t), u(t))\, dt,$$

by specifying the functions g and h as

$$g(x, u) = 1 + x^T Q x + u^T R u, \tag{5.1}$$

$$h(x) = x^T Q_f x, \tag{5.2}$$

i.e., a combined cost of the terminal time and quadratic state/input penalties. Our framework supports more general expressions for cost functionals J, but discussions in this chapter are focused on the terminal time and quadratic penalties by (5.1)-(5.2).

Moreover, we restrict the dynamical system (1.2)

$$\dot{x}(t) = f(x(t), u(t)), \quad x(t) \in X_{\text{free}} \cap X_{\text{feas}},\ u(t) \in U_{\text{feas}},$$

into a controllable linear system [53]

$$\dot{x} = Ax(t) + Bu(t), \tag{5.3}$$

where the $n \times n$ matrix A and the $m \times m$ matrix B together satisfy the condition

$$\operatorname{rank}\left(\begin{bmatrix} B & AB & A^2B & \ldots & A^{n-1}B \end{bmatrix}\right) = n. \tag{5.4}$$

We emphasized in Chapter 1 that any optimal solution in the literature, except analytic optimal solutions, relaxes the optimality definition. Such a relaxation often comes with discretization, numerical integration, basis functions, and others. We define the optimality property we achieve in this chapter as follows:

Definition 5.1 (Optimality of Concatenated Trajectory) *Within a set of trajectories defined by maximum time duration ΔT of segments and n extra basis functions, we call a trajectory is ΔT-optimal with n-DoF (Degree of Freedom) if no better trajectory can be concatenated with segments longer than or equal to ΔT.*

For more succinct discussions, we define the small-time reachable set as follows:

Definition 5.2 (Small-Time Reachable Set) *Let $\mathcal{R}(z, t_f)$ denote a forward small-time reachable set from a state z within time t_f, and let $\mathcal{R}(z, -t_f)$ denote a backward small-time reachable set to z within t_f:*

$$\mathcal{R}(z, t_f) = \{x(t) | (\mathbf{x}, \mathbf{u}, t) \in \mathcal{T},\ t \le t_f,\ x(0) = z\}, \tag{5.5}$$

$$\mathcal{R}(z, -t_f) = \{x(0) | (\mathbf{x}, \mathbf{u}, t) \in \mathcal{T},\ t \le t_f,\ x(t) = z\}. \tag{5.6}$$

In this chapter, we often utilize that $\mathcal{R}(z, t_f)$ with a certain t_f value or larger becomes a superset of the forward `NearNeighbors` *procedure, if the nearness in the* `NearNeighbors` *procedure is measured by the pseudo-metric $\bar{J}$ defined in Chapter 1. Similarly, $\mathcal{R}(z, -t_f)$ may be used as a superset of the backward near neighbors.*

5.3 Flatness-Based Local Steering Algorithm

5.3.1 Differential Flatness

A nonlinear system $\dot{x} = f(x, u)$, $x \in \mathbb{R}^n$, $u \in \mathbb{R}^m$ is called differentially flat [35] if there exists a *flat output*

$$z = \Psi_z(x, u, \dot{u}, \dots, u^{(\ell)}), \quad z \in \mathbb{R}^m \tag{5.7}$$

such that $x = \Psi_x(z, \dot{z}, \ldots, z^{(p)})$, $u = \Psi_u(z, \dot{z}, \ldots, z^{(q)})$, where $z^{(k)}$ denotes the k-th derivative of z. The definition implies that trajectories of $\bar{z} := (z, \dot{z}, \ldots, z^{(r)})$, a tuple of the flat output z and its derivatives where $r = \max(p, q)$, are mapped into trajectories of x and u via the maps Ψ_x and Ψ_u. Hence, a trajectory of state x obtained via the map Ψ_x is realizable by executing a trajectory of input u obtained via the map Ψ_u, given a desired trajectory of the flat output $\bar{z}$.

Controllable linear systems with single input are flat, as the state-space representation in *controllable canonical form* immediately affirms. With multiple inputs, a system can be linearly transformed and decomposed into blocks of multiple subsystems. Diagonal blocks represent subsystems in single-input controllable canonical forms, and non-diagonal blocks weakly connect such subsystems [77]. By selecting flat outputs for subsystems in the same manner with the single-input cases, multi-input controllable linear systems are flat as well. Nilpotent or not, examples in [117] of linear systems are flat.

Remark 5.3 *Controllable linear systems are flat. Flat outputs are identified as least-differentiated variables in controllable canonical forms. Refer [77] for details in multi-input cases.*

In illustrative examples below, flat outputs are $z = \theta$ in linearized pendulum (5.8) and $z = (x, y)$ in 2D double integrators (5.9). Note that four blocked subsystems in (5.9) are decoupled by null matrices in non-diagonal blocks, but subsystems are weakly connected in general cases.

$$\begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ \frac{g}{\ell} & -\frac{b}{ml^2} \end{bmatrix} \begin{bmatrix} \theta - \pi \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{ml^2} \end{bmatrix} u, \tag{5.8}$$

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{y} \\ \ddot{y} \end{bmatrix} = \left[\begin{array}{cc|cc} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{array}\right] \begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{bmatrix} + \left[\begin{array}{c|c} 0 & 0 \\ 1 & 0 \\ \hline 0 & 0 \\ 0 & 1 \end{array}\right] \begin{bmatrix} u_x \\ u_y \end{bmatrix}. \tag{5.9}$$

5.3.2 State and Input Constraints

For a trajectory to be certified as dynamically feasible, imposed state/input constraints need to be satisfied at all points along the trajectory. In the literature, a canonical way to enforce state/input constraints is adding the state/input constraints at dense collocation points as constraints in the optimization problem [79, 32]. Assuming the resolution of collocated points is set to be fixed as a small number, the approach implies that local steering with a larger terminal time tends to contain a larger number of constraints in optimization.

In our approach, the best trajectory is obtained tentatively, by optimizing the cost functional with active constraints only. If the tentative trajectory violates any state/input constraints, additional constraints are added progressively to the optimization problem at the point of the locally-maximal violation for constraints. The loop of progressive constraints returns success if the tentative trajectory never violates a constraint, or failure if the active constraints lead to infeasible problems. Section 5.3.3 provides more detailed implementation of progressive constraints.

5.3.3 Local Steering Given a Terminal Time

Standard local steering methods for flat systems involve basis functions [111, 79, 32, 78]. For the obtained solution to satisfy boundary conditions, the number of bases is at least the same as the number of boundary conditions. Typically, optimization-centric approaches increase the number of extra bases significantly. In our approach, the number of free coefficients for extra bases (or DoF) n is set to be small in the beginning, and the number may be incrementally or selectively increased for refinement purposes. Note that the setup is analogous to finding a (n-1)th-order hold control input u that reaches the final state at the terminal time, where the exact recovery of the state is hard to achieve [51]. On the contrary, our approach with basis functions exactly recovers the states at boundaries, while state and input constraints along the trajectory need to be checked carefully.

For brevity and clarity, we describe the exemplary procedure for a single-input system with 1-DoF polynomial basis functions, and add general statements. In an ℓ-dimensional space of a flat output z and its derivatives $\bar{z} = (z, \dot{z}, \dots, z^{(\ell-1)})$, the minimal number of bases required to connect two states $\bar{z}(0)$ and $\bar{z}(T)$ is $N = 2\ell$, given a terminal time T. We employ

$$z(t) = \sum_{i=0}^{2\ell} a_i t^i, \quad \text{for } t \in [0, T], \tag{5.10}$$

with $N = 2\ell + 1$ polynomial bases, where $a_{2\ell}$ is the free coefficient without loss of generality. Boundary conditions

$$i!\, a_i = z^{(i)}(0), \quad \text{for } i = 0, \dots, \ell - 1, \tag{5.11}$$

at $t = 0$ immediately determine first ℓ coefficients, and an invertible (guaranteed if $T \neq 0$) matrix equation

$$\begin{bmatrix} T^{2\ell-1} & \cdots & T^{\ell} \\ \vdots & \ddots & \vdots \\ \frac{(2\ell-1)!}{\ell!} T^{\ell} & \cdots & \frac{\ell!}{1!} T^{1} \end{bmatrix} \begin{bmatrix} a_{2\ell-1} \\ \vdots \\ a_{\ell} \end{bmatrix} = \begin{bmatrix} z(T) - a_{2\ell} T^{2\ell} - \sum_{i=0}^{\ell-1} a_i T^i \\ \vdots \\ z^{(\ell-1)}(T) - \frac{(2\ell)!}{(\ell+1)!} a_{2\ell} T^{\ell+1} - \sum_{i=\ell-1}^{\ell-1} \frac{(i)!}{1!} a_i T^{i-\ell+1} \end{bmatrix} \tag{5.12}$$

follows from boundary conditions at $t = T$. With $a_{2\ell}$ as the variable to be determined, (5.12) turns into functions of $a_{2\ell}$

$$a_i = f_{a,i}(a_{2\ell}; T, \bar{z}(0), \bar{z}(T)), \quad \text{for } i = \ell, \dots, 2\ell - 1, \tag{5.13}$$

where $f_{a,i}$ is linear in $a_{2\ell}$. For an increased DoF n, each function $f_{a,i}$ is still linear in free coefficients $a_{2\ell}, \dots, a_{2\ell+n-1}$.

Each linear state/input constraint $c(x,u) \leq 0$ at $t = t_r T$, $0 \leq t_r \leq 1$ leads to

$$\begin{aligned} c(x,u) &= c(\Psi_x(\bar{z}(t_r T)), \Psi_u(\bar{z}(t_r T))) = \Psi_c\left(a_{2\ell}, f_{a,\ell}, \ldots, f_{a,2\ell-1}; t_r, T, \bar{z}(0), \bar{z}(T)\right) \\ &= \Psi_c\left(a_{2\ell}; t_r, T, \bar{z}(0), \bar{z}(T)\right) \leq 0, \end{aligned} \tag{5.14}$$

where Ψ_c is also a linear mapping. The time ratio

$$t_r = \frac{1}{T} \underset{t_a T \leq t \leq t_b T}{\operatorname{argmax}} \, c(\Psi_x(\bar{z}(t)), \Psi_u(\bar{z}(t))) \tag{5.15}$$

of locally maximal violation is identified by root-finding in (2ℓ-1)th or smaller order polynomials and the constraint at $t = t_r T$ is added to the optimization formulation over $a_{2\ell}$. The intersection of constraints defines the feasible region of $a_{2\ell}$ as an interval. For an increased DoF n, progressive constraints are linear in free coefficients $a_{2\ell}, \ldots, a_{2\ell+n-1}$, and defines the feasible region as a convex polytope.

In our choice of the functions g and h in (5.1)-(5.2), we allow the symmetric matrices Q and R to be zero, as a special case for pure minimum-time problems. Otherwise, positive-semidefinite Q and positive-definite R matrices are required. After symbolic integration over time $[0, T]$, J becomes a quadratic function of $a_{2\ell}$ as follows:

$$\begin{aligned} J &= \Psi_J\left(a_{2\ell}, f_{a,\ell}, \ldots, f_{a,2\ell-1}; T, \bar{z}(0), \bar{z}(T)\right) \\ &= \Psi_J\left(a_{2\ell}; T, \bar{z}(0), \bar{z}(T)\right) \end{aligned} \tag{5.16}$$

The 1-DoF optimization result can be algebraically calculated over the feasible interval of $a_{2\ell}$. For an increased DoF n or multi-input cases, Ψ_J still remains as quadratic over free coefficients $a_{2\ell}, \ldots, a_{2\ell+n-1}$ or across free coefficients $a_i, b_i, \ldots$ from other flat outputs, thus leading to a quadratic programming problem subject to linear constraints obtained in (5.14). Let $\bar{x} = (a_{2\ell}, \ldots, a_{2\ell+n-1}, 1)$ denote a vector of free coefficients and a constant 1, then the overall problem is

$$\text{minimize } \bar{x}^T S \bar{x}, \text{ s.t. } C\bar{x} \leq D \tag{5.17}$$

where n, T, $\bar{z}(0)$, $\bar{z}(T)$ determine the positive-definite matrix S, and progressively added constraints (5.14) determine C and D matrices. For min-time problems, only linear programming problems are needed to check the constraints along the trajectory.

```
1 $c \leftarrow$ AddConstraints($\bar{z}_0, \bar{z}_f, T, \{0,1\} \cup$ Violated($\bar{z}_0, \bar{z}_f, T, e$));
2 while Feasible($c$) do
3     $(e, J) \leftarrow$ Optimize($c, e$);
4     if $t_r \leftarrow$ Violated($\bar{z}_0, \bar{z}_f, T, e) = \emptyset$ then
5         return $(\{\bar{z}_0, \bar{z}_f, T, e\}, J)$;
6     else
7         $c \overset{\cup}{\leftarrow}$ AddConstratints($\bar{z}_0, \bar{z}_f, T, t_r$);
8 return $(\{\bar{z}_0, \bar{z}_f, \infty, \emptyset\}, \infty)$;
```

Algorithm 5.1: TPBVP Given T ($\bar{z}_0$, $\bar{z}_f$, T, $(e = \emptyset)$)

Algorithm 5.1 summarizes the fixed-time local steering between two states $\bar{z}_0$ and $\bar{z}_f$, while the outer loop Algorithm 5.2 specifies the arrival time T. Let e denote the set of free coefficients $a_{2\ell}, \ldots, a_{2\ell+n-1}$, or the edge that contains sufficient information to connect two states given boundary conditions and arrival time. For the designated free coefficients e, the `Violated` procedure detects t_r values at which constraints are locally violated the most. The `AddConstraints` procedure adds constraints at t_r values. The algorithm returns the best found cost and sufficient information to recover such a trajectory.

Proposition 5.4 *Suppose a single-input linear system, a terminal time T, linear state/input constraints, and $2\ell+n$ (or n-DoF) polynomial bases. Checking constraints at $t = t_r T$, $0 \leq t_r \leq 1$ is equivalent to root-finding in $(2\ell + n - 2)$th or smaller order polynomials. Quadratic programming problems over free coefficients $a_{2\ell}, \ldots, a_{2\ell+n-1}$ contain n variables to optimize. For multi-input cases, the number of variables in optimization increases accordingly, while the complexity of checking constraints is unchanged.*

Remark 5.5 *Let $a_{2\ell}, \ldots, a_{2\ell+n_0-1}$ be a local steering solution with n_0-DoF. Then, any local steering with DoF $n > n_0$ has a feasible solution based on $a_{2\ell}, \ldots, a_{2\ell+n_0-1}$, by setting additional coefficients $a_{2\ell+n_0}, \ldots, a_{2\ell+n-1} = 0$. Thus, warm start with feasible solutions is guaranteed when refining solutions with larger DoFs.*

In checking the feasibility of linear constraints, we exploit *dual* formulations for *primal* problems with pivoting methods such as *simplex algorithm* [14]. Especially with larger number of constraints than DoFs, incremental addition of constraints is handled efficiently. Specifically, for the pair

$$\begin{aligned} &\text{minimize} \quad c'x && \text{maximize} \quad p'b \\ &\text{s.t.} \quad Ax \geq b, && \text{s.t.} \quad p'A = c' \\ & && \qquad\quad p \geq 0, \end{aligned} \tag{5.18}$$

of primal and dual problems, dual indices for *basic variables* by the previous iteration remain basic after progressively adding more constraints by (5.14). Thus, the current computation effectively continues from the last effort. In addition, by the duality theory [14], a feasible dual solution and nonzero dual variables lead to a primal solution, obtained by active primal constraints corresponding to dual indices. Unbounded dual solutions inform the infeasibility of primal problems. Infeasible dual solutions suggest unbounded or infeasible primal problems, but the situation is hardly relevant to our formulation with more primal constraints than variables.

5.3.4 Local Steering with the Terminal Time Search

The fixed-time local steering (Algorithm 5.1) needs to be encapsulated by a search procedure (Algorithm 5.2) for the terminal time. Previous work is either limited to fixed (or infinite) time scenarios [89, 41, 111, 79, 32] or subject to time search [51, 117, 43, 49, 78] for generality. We efficiently shrink the search range by informing and updating the bound, as introduced in the collective steering scheme (Algorithm 2.22).

Lower and upper bounds for time search, T_L and T_U, can be computed by exploiting the non-negativity of g in (5.1), boundary conditions, state/input constraints, and costs of any or best found local steering solution. Accordingly, the `TimeRange` procedure in Line 2 and 7 finds and updates the range $[T_\mathrm{L}, T_\mathrm{U}]$, and effectively decreases the portion of unrewarding attempts for fixed-time local steering. Empirically, the search resolution dT can be chosen crudely and further refined if promising. For a lengthy trajectory, Line 8 checks the time duration, divides into smaller ones, and attempts to refine each segment by calling the Algorithm 5.2 again, with e_seg as the upper-bound edge upon no success in refinement. As a result, time duration of any edge is shorter than or equal to ΔT, and our algorithm achieves the following property.

```
1  $J_\mathrm{best} \leftarrow \infty$;
2  $(T_\mathrm{L}, T_\mathrm{U}) \leftarrow$ TimeRange($\bar{z}_0, \bar{z}_f, e_\mathrm{best}, J_\mathrm{upper}$);
3  for $T \in [T_\mathrm{L} : dT : T_\mathrm{U}]$ do
4      $(e, J) \leftarrow$ Algorithm5.1($\bar{z}_0, \bar{z}_f, T$);
5      if $J < \min(J_\mathrm{upper}, J_\mathrm{best})$ then
6          $e_\mathrm{best} \leftarrow e$;   $J_\mathrm{best} \leftarrow J$;
7          $(T_\mathrm{L}, T_\mathrm{U}) \leftarrow$ TimeRange($\bar{z}_0, \bar{z}_f, e_\mathrm{best}, J_\mathrm{best}$);
8  if $\Delta T < T(e_\mathrm{best})$ then
9      $(\bar{Z}_0, \bar{Z}_f, J_\mathrm{seg}, E_\mathrm{seg}) \leftarrow$ DivideTrajectory($e_\mathrm{best}$);
10     $(e_\mathrm{refine}, J_\mathrm{refine}) \leftarrow (\emptyset, 0)$;
11     for $\bar{z}_{s0} \in \bar{Z}_0,\ \bar{z}_{sf} \in \bar{Z}_f,\ J_\mathrm{s} \in J_\mathrm{seg},\ e_\mathrm{seg} \in E_\mathrm{seg}$ do
12         $(e_\mathrm{refine}, J_\mathrm{refine}) \overset{+}{\leftarrow}$ Algorithm5.2($\bar{z}_{s0}, \bar{z}_{sf}, J_\mathrm{s}, e_\mathrm{seg}$);
13     return $\min\left((e_\mathrm{best}, J_\mathrm{best}), (e_\mathrm{refine}, J_\mathrm{refine})\right)$;
14 return $(e_\mathrm{best}, J_\mathrm{best})$
```

Algorithm 5.2: TPBVP ($\bar{z}_0$, $\bar{z}_f$, J_upper, $(e_\mathrm{best} = \emptyset)$)

Proposition 5.6 *Assume RRT* or GR-FMTs employ Algorithm 5.2 for the TPBVP. Then, the best found trajectory approaches n-DoF ΔT-optimal trajectory in the limit.*

Note that division and refinement of a lengthy trajectory do not immediately construct a n-DoF ΔT-optimal trajectory between two states. Instead, outer loop attempts and finds better concatenation of trajectory segments in the limit.

The threshold ΔT for trajectory division matters in early iterations, and later iterations generate trajectories shorter than ΔT. Thus, revisiting and refining edges with an increased DoF or decreased ΔT provably improve the quality of solutions by enhancing the defined class of optimality, but with increased computation. Our current implementation does not change DoFs and ΔT.

5.4 Simulation Results

Our simulation is based on a laptop with Intel® Core™ i5-4200U (1.60GHz) processor and 4GB RAM. For all examples, although feedback motion trees are constructed, only selected trajectories are displayed for clear visualization.

5.4.1 Constrained 1D Double Integrator in Free Space

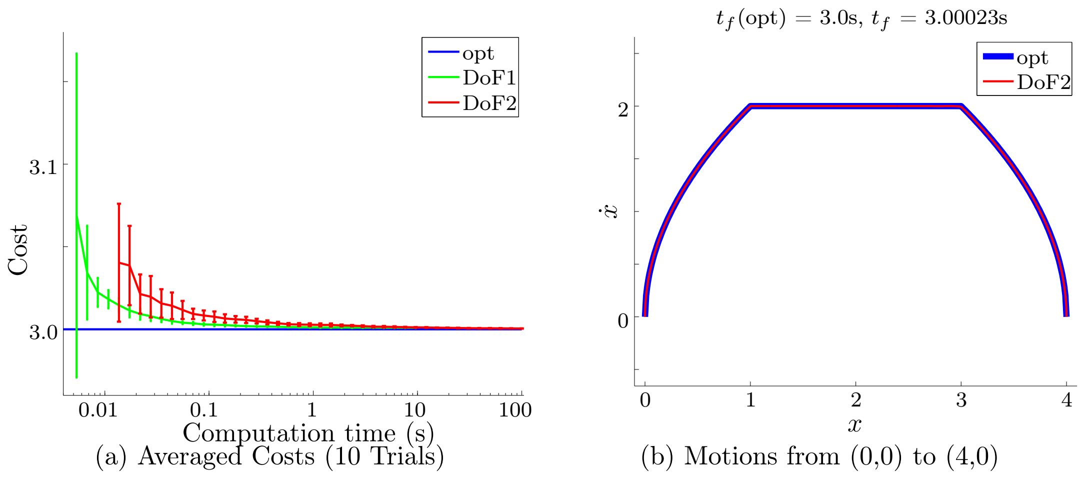
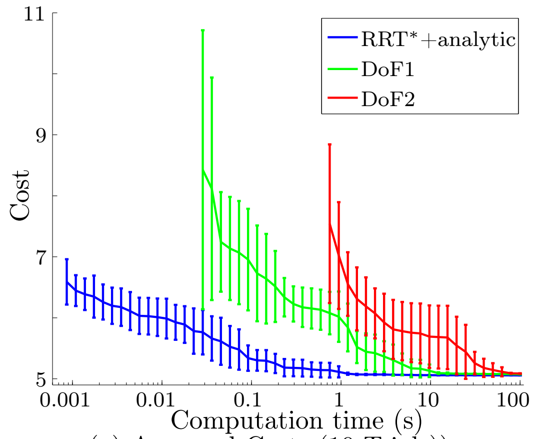


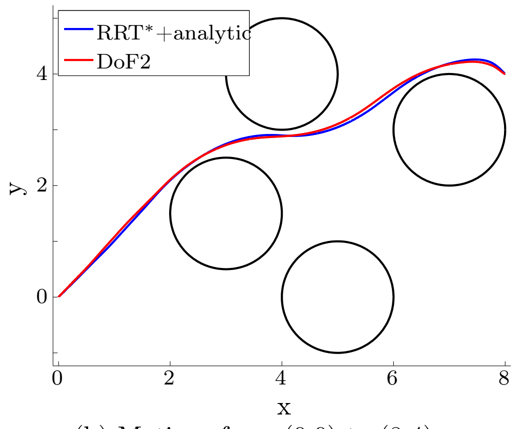
Figure 5-1: 1D min-time double integrator. GR-FMTs with 2-DoF (red) and ΔT=0.2 shortly generate a comparable motion to analytic solution (blue).

Figure 5-1 shows min-time trajectories of a double integrator with constraints $|\dot{x}| \leq 2$, $|\ddot{x}| \leq 2$. Note that analytic optimal solutions, if available, are always better and faster than others, and the comparison is only to demonstrate our comparable results in an anytime fashion [55] that finds the first solution in a short time and improves the best found solution to optimality given time. For 1 and 2 DoFs, TPBVP was called 13±2 and 0.18±0.008 million times in parts of 10 seconds.

5.4.2 Constrained/Obstructed 2D Double Integrators



(a) Averaged Costs (10 Trials))



(b) Motions from (0,0) to (8,4)

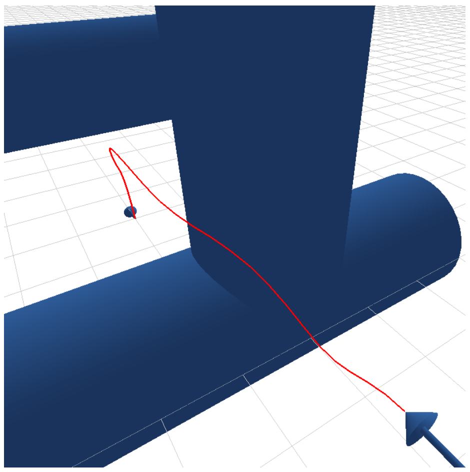
Figure 5-2: 2D min-time double integrators, from the RRT* using analytic solutions [56] (blue) and GR-FMTs (green, red) with DoFs and ΔT=0.5.

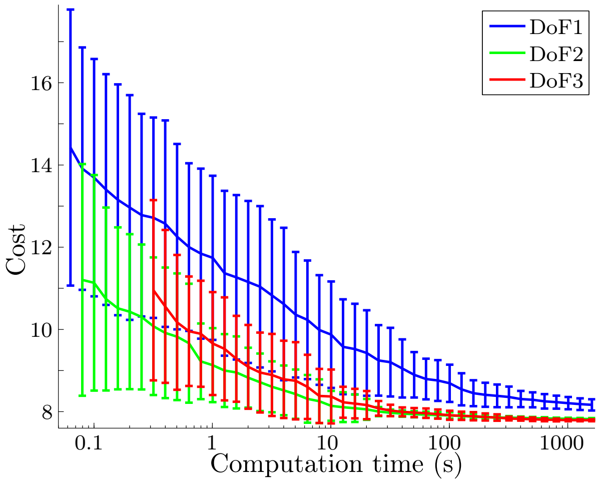
For min-time 2D double integrators (5.9), the RRT* with analytic solutions [56] provides asymptotically optimal results. With constraints $|\dot{x}| \leq 2$, $|\dot{y}| \leq 2$, $|\ddot{x}| \leq 2$, $|\ddot{y}| \leq 2$, Figure 5-2 compares our approach to the approach in [56]. Remind that analytic optimal solutions, if available, are always faster and better, while our approach is general to controllable linear systems (without analytic solutions) in an anytime fashion. For analytic solutions and 1-to-3 DoFs, TPBVP was called 102±10, 48±5, 1.8±0.09, 0.66±0.08 million times in parts of 100 seconds.

5.4.3 Landing of Constrained/Obstructed Helicopter

Minimum-time helicopter landing using a linearized model

$$\begin{aligned}
\ddot{x} &= -b_x\dot{x} - g\theta, \quad \ddot{\theta} = -2\zeta_\theta\omega_\theta\dot{\theta} - \omega_\theta^2\theta + \omega_\theta^2\theta_{\text{ref}},\\
\ddot{y} &= -b_y\dot{y} + g\phi, \quad \ddot{\phi} = -2\zeta_\phi\omega_\phi\dot{\phi} - \omega_\phi^2\phi + \omega_\phi^2\phi_{\text{ref}},\\
\ddot{z} &= -\mu\dot{z} + \mu\omega_{\text{ref}}, \qquad (5.19)
\end{aligned}$$

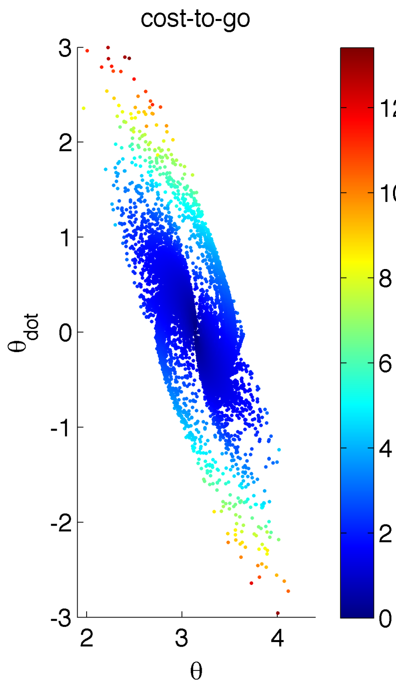

(a) Landing Trajectory



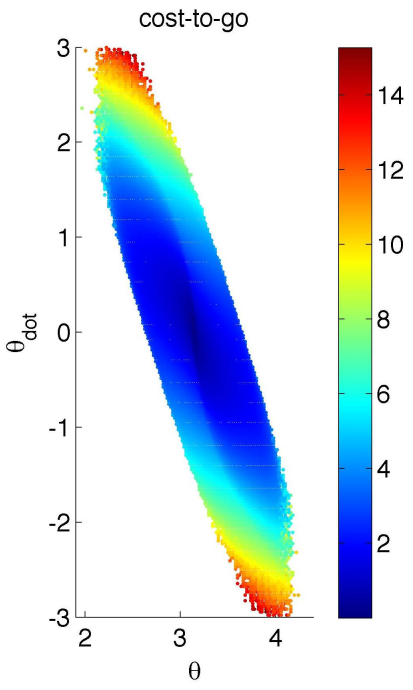
(b) Averaged Costs (20 Trials)

Figure 5-3: Min-time landing of helicopter by GR-FMTs with DoFs and $\Delta T = 0.5$

is shown in Figure 5-3, with bounded θ_{ref}, ϕ_{ref}, ω_{ref}. Expected trade-offs by DoFs are clear for computation and asymptotic solution quality. For 1 to 3 DoFs, TPBVP was called 51±6, 29±2, 5±1 million times in parts of 1,600 seconds.



(a) Tree (70064 Vertices)



(b) Computed on Grids

Figure 5-4: For a torque-limited linearized pendulum around $(\pi, 0)$, cost-to-go values $J = \int 1 + \ddot{\theta}^2\, dt$ are color-coded on vertices of GR-FMTs and grids.

5.4.4 Torque-Limited Inverted Pendulum

For a linearized pendulum (5.8) with $|u| \leq u_{\max}$, Figure 5-4 shows GR-FMTs with 1-DoF and ΔT=0.15 that minimize $J = \int 1 + \ddot{\theta}^2 \, dt$. GR-FMTs do not require grids, and cost-to-go values are shown for visualization purposes only. Obtained feedback policies are comparable to the constrained LQR [11], but with more generality toward obstructed scenarios.

5.5 Conclusions

The application areas of the RRT* and GR-FMTs are largely expanded or limited by the performance of the associated local steering methods. It may be surprising that analytic solutions for local steering — the fastest possible form of local steering solutions — are unavailable for controllable linear systems with linear state/input constraints. For the problem, our proposed computation of a relaxed local steering remains light, and it leads to asymptotically ΔT-optimal solutions with DoFs. In the presence of obstacles with unspecified or uncertain computational budgets, our local steering approach embedded in an anytime planner such as the RRT* or GR-FMTs has clear advantages over other approaches.

Chapter 6

Conclusions and Remarks

In this dissertation, we have presented a set of contributions to sampling-based motion planning algorithms. The first part (Chap 2) consists of a review for the state of the art, a proposed generalization of optimal sampling-based motion planning algorithms including modifications aimed at improving the performance, and the ability to handle dynamical systems more efficiently. The second part (Chap 3, 4, and 5) developed subroutines that are essential to sampling-based motion planning for dynamical systems, i.e., efficient solution approaches to Two-Point Boundary Value Problems (TPBVPs). Specifically, TPBVP solution approaches were presented for three classes of dynamical systems with demonstrating examples.

In this chapter, we first summarize the contributions of this dissertation, and then discuss promising ideas for future research directions.

6.1 Summary

In Chap 2, we started with an extensive literature review for sampling-based motion planning algorithms that equip with optimality guarantees. Such algorithms were reviewed and interpreted within an abstract algorithmic framework, in order to allow component-wise comparisons of improved algorithms. Key ideas of the compared algorithms were adapted into incremental procedures that are suitable with the RRT*, and we presented our enhanced RRT* by integrating the discussed adaptations to the

RRT*. The enhanced RRT*, an incremental algorithm that enhances the RRT*, substantially improved the convergence rates to the optimum, especially within the strict $\mathcal{O}(\log n)$ complexity per iteration.

The second focus of Chap 2 was on feedback motion planning, based on the tree-based incremental sampling-based algorithms we had considered. The modification of the forward tree-based algorithm into the backward algorithm, accompanied with the local connection attempts to $\mathcal{O}(\log n)$ neighbors only, allows efficient replanning and feedback planning with guarantees on the asymptotic optimality of the feedback policies and the expected closed-loop trajectories. By design, the proposed feedback planning algorithm efficiently maintains and reuses substantial parts of the previous computation.

The third focus of Chap 2 was the re-design for the loop of TPBVPs, in order to efficiently handle computationally expensive TPBVPs for dynamical systems. Largely inspired by the well-known branch and bound technique, our re-designed loop of cost-bounded TPBVPs significantly reduced the overall computation required for TPBVPs, thus enabled the consideration of more complicated dynamical systems for motion planning, while using the same amount of computation.

In Chap 3, we started the discussion of TPBVP solution approaches, by proposing a numerical method that utilizes the influence of control inputs to states in a lower-dimensional task space. Time-optimal off-road vehicle maneuvers were considered as the main examples for aggressive cornering scenarios with several turning degrees. The results in this chapter demonstrated and suggested the necessity of computationally efficient TPBVP solutions that exactly connect two states.

In Chap 4, we proposed a semi-analytic TPBVP solution for pseudo-flat dynamical systems. By identifying a suitable change of coordinates that transforms the system into a pseudo-flat system, computation of the time-optimal off-road vehicle maneuvers was expedited. As a result, more complicated scenarios could be considered, e.g., for closed race tracks. This chapter demonstrated the benefits of mapping constraints to the flat output space.

In Chap 5, the idea of mapping constraints to the flat output space was deepened for a subset of differentially flat systems, i.e., controllable linear systems. The mappings from constraints, constraint violations, and the cost functional to the flat output space transformed the TPBVP into an incremental computation of small-sized linear or quadratic programming problems, subject to progressively added constraints. Combined with the feedback planning algorithm in Chap 2, the TPBVP approach could generate asymptotically optimal feedback policies for linear systems, with guarantees for the collision-free trajectories.

6.2 Future Research Directions

In this section, we discuss promising directions for further research and development.

6.2.1 Integration of the Proposed Ideas

The implementation and the simulation examples in Chap 2 mainly focused on fair comparisons among key ideas that accelerate the solution's convergence to the optimum. Such algorithmic modifications as a whole, or at least the enhanced RRT* algorithm for trivial systems (straight path segments), may be potentially provided to the general public as an open source library.

In addition, the implementation and the simulation results in Chap 3, 4, and 5 actually precede the inception of most ideas in Chap 2, as hidden driving forces to develop and elaborate the speed-up ideas in Chap 2. From the practical point of view, applying the entire set of ideas in Chap 2 back to the implementation in Chap 3, 4, and 5 has great potentials for improved computational performance.

6.2.2 Generalization of TPBVP Approaches

TPBVP solution approaches in Chap 3, 4, and 5 may be further generalized. This dissertation described the solution approaches based on examples, thus further abstraction may prevent trials and errors by readers for other extended examples.

First, Chap 3 and 4 presented TPBVP solutions for a specific dynamical system, i.e., half-car model. The numerical method for dynamical systems and the semi-analytic method for pseudo-flat systems could be described as a principled procedure with more generality. Perhaps, the procedure could be generalized, but only based on the categorization of dynamical systems.

Second, Chap 5 presented TPBVP solutions for linear systems, using polynomial basis functions of time. Other types of basis functions may be considered and compared for better performance and more suitable computation, especially in searching for the terminal time of the trajectory segment.

Third, the approach in Chap 5 utilized the differential flatness, by focusing on linear systems, of which the constraints in the flat output space could be represented and managed suitably. The approach may be attempted for an extension to differentially flat systems, although the inclusion of constraints in the problem formulation becomes more challenging. Another type of potential extension would be based on the successive approximation of general dynamical systems using flat or linear systems.

6.2.3 Further Specification and Characterization of Dynamics

Chap 4 utilized the characteristics of the considered dynamical system, i.e., the center of oscillation for the coordinate transformation. Further specification for dynamical systems, as opposed to the generalization suggested in Chap 6.2.2, typically contributes to more efficient algorithms or computation for TPBVPs, by coordinate transformation, simplifying assumptions, simplifying control inputs, and so forth.

6.2.4 Exact TPBVPs to Inexact Local Steering Problems

Chap 3 briefly mentioned an alternative solution approach to ignore motion gaps in the tree or delay the repropagation procedure that eliminates the motion gaps. By relaxing the TPBVP that exactly connects two points into the local steering problem that does not necessarily require an exact connection of two points, a larger set of approaches and modified methods can be attempted for local steering problems and sampling-based algorithms. One of the prerequisites for inexact methods is the mathematically correct way to estimate the region in which the connection can be considered to be exact, or to estimate the connection cost that accounts for the remaining motion gap.

6.2.5 Deterministic to Stochastic

The algorithms in this dissertation are currently applicable to deterministic models, which may not be realistic in the scenario with autonomous vehicles on unknown terrain models. Future work would involve extending the current approach to handle uncertainties and modeling errors. As the majority of methods for stochastic systems internally employ deterministic solutions as the starting or guiding points, further extension to stochastic methods may be naturally made.

6.2.6 Development of A Vehicle

Each chapter in this dissertation was deeply motivated by examples of autonomous ground vehicles. The development of an autonomous ground vehicle and the validation of the proposed algorithms in this dissertation on the vehicle would be a natural procedure to be followed.

We had assumed that the cost functional to minimize was known for motion planning. In the actual vehicle's case, identifying the passenger's preference as the cost functional would become an important procedure beforehand.

THIS PAGE INTENTIONALLY LEFT BLANK

Bibliography

[1] Jürgen Ackermann. Robust decoupling, ideal steering dynamics and yaw stabilization of 4WS cars. *Automatica*, 30(11):1761–1768, 1994.

[2] Baris Akgun and Mike Stilman. Sampling heuristics for optimal motion planning in high dimensions. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, September 2011.

[3] Ron Alterovitz, Sachin Patil, and Anna Derbakova. Rapidly-exploring roadmaps: Weighing exploration vs. refinement in optimal motion planning. In *IEEE International Conference on Robotics and Automation*, May 2011.

[4] Nancy M. Amato and Guang Song. Using motion planning to study protein folding pathways. *Journal of Computational Biology*, 9(2):149–168, 2002.

[5] Nancy M. Amato and Yan Wu. A randomized roadmap method for path and manipulation planning. In *IEEE International Conference on Robotics and Automation*, 1996.

[6] Oktay Arslan and Panagiotis Tsiotras. Use of relaxation methods in sampling-based algorithms for optimal motion planning. In *IEEE International Conference on Robotics and Automation*, May 2013.

[7] Eugene Asarin, Thao Dang, Goran Frehse, A. Girard, Colas Le Guernic, and Oded Maler. Recent progress in continuous and hybrid reachability analysis. In *Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control, 2006 IEEE*, pages 1582–1587, October 2006.

[8] E. Bakker, L. Nyborg, and H. B. Pacejka. Tyre modelling for use in vehicle dynamics studies. In *SAE International Congress and Exposition*, 1987.

[9] Efstathios Bakolas and Panagiotis Tsiotras. Optimal synthesis of the asymmetric sinistral/dextral Markov-Dubins problem. *Journal of Optimization Theory and Applications*, 150(2):233–250, 2011.

[10] Richard Bellman. *Dynamic Programming.* Princeton University Press, 1957.

[11] Alberto Bemporad, Manfred Morari, Vivek Dua, and Efstratios N. Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20, 2002.

[12] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[13] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control.* Athena Scientific, 1995.

[14] Dimitris Bertsimas and John N. Tsitsiklis. *Introduction to Linear Optimization.* Athena Scientific, 1997.

[15] Amit Bhatia and Emilio Frazzoli. Incremental search methods for reachability analysis of continuous and hybrid systems. In *Hybrid Systems: Computation and Control*, volume 2993 of *Lecture Notes in Computer Science*, pages 142–156. Springer-Verlag, 2004.

[16] Joshua John Bialkowski. *Optimizations for sampling-based motion planning algorithms.* PhD thesis, Massachusetts Institute of Technology, February 2014.

[17] Paul T. Boggs and Jon W. Tolle. Sequential quadratic programming. *Acta numerica*, 4:1–51, 1995.

[18] Robert Bohlin and Lydia E. Kavraki. Path planning using lazy prm. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 521–528 vol.1, April 2000.

[19] Bela Bollobas and Oliver Riordan. *Percolation.* Cambridge University Press, 2006.

[20] Arthur E. Bryson, Jr. Optimal control - 1950 to 1985. *IEEE Control Systems Magazine*, 16:26–33, Jun 1996.

[21] Arthur E. Bryson, Jr. and Yu-Chi Ho. *Applied Optimal Control.* Hemisphere Publishing Corporation, 1975.

[22] Eduardo F. Camacho and Carlos Bordons. *Model Predictive Control.* Springer, 1999.

[23] D. Casanova, R. S. Sharp, and P. Symonds. Minimum time manoeuvring: The significance of yaw inertia. *Vehicle System Dynamics*, 34:77–115, 2000.

[24] Hsuan Chang and Tsai-Yen Li. Assembly maintainability study with motion planning. In *IEEE International Conference on Robotics and Automation*, 1995.

[25] Peng Cheng and Steven M. LaValle. Resolution complete rapidly-exploring random trees. In *IEEE International Conference on Robotics and Automation*, 2002.

[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] Raghvendra .V. Cowlagi and Panagiotis Tsiotras. Hierarchical motion planning with dynamical feasibility guarantees for mobile robotic vehicles. *Robotics, IEEE Transactions on*, 28(2):379–395, 2012.

[28] K. Daniel, A. Nash, S. Koenig, and A. Felner. Theta*: Any-angle path planning on grids. *Journal of Artificial Intelligence Research*, 39:533–579, 2010.

[29] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[30] B. Donald, P. Xavier, J. Canny, and J. Reif. Kinodynamic motion planning. *Journal of the ACM*, 40:1048–1066, November 1993.

[31] 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*, 79:497–516, 1957.

[32] Nadeem Faiz, Sunil K. Agrawal, and Richard M. Murray. Trajectory planning of differentially flat systems with dynamics and inequalities. *Journal of Guidance, Control, and Dynamics*, 24(2):219–227, 2001.

[33] P. Ferbach. A method of progressive constraints for nonholonomic motion planning. In *IEEE International Conference on Robotics and Automation*, 1996.

[34] D. Ferguson, N. Kalra, and A. Stentz. Replanning with RRTs. In *IEEE International Conference on Robotics and Automation*, Orlando, FL, May 2006.

[35] Michel Fliess, Jean Lévine, Philippe Martin, and Pierre Rouchon. Flatness and defect of non-linear systems: introductory theory and examples. *International Journal of Control*, 61(6):1327–1361, 1995.

[36] Michel Fliess, Jean Lévine, Philippe Martin, and Pierre Rouchon. A Lie-Bäcklund approach to equivalence and flatness of nonlinear systems. *Automatic Control, IEEE Transactions on*, 44(5):922–937, 1999.

[37] Emilio Frazzoli, Munther A. Dahleh, and Eric Feron. Real-time motion planning for agile autonomous vehicles. *AIAA Journal of Guidance, Control, and Dynamics*, 25:116–129, 2002.

[38] Stefan Fuchshumer, Kurt Schlacher, and Thomas Rittenschober. Nonlinear vehicle dynamics and control – a flatness based approach. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 6492–6497, Seville, Spain, December 12–15 2005.

[39] Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, September 2014.

[40] Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. Batch Informed Trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *IEEE International Conference on Robotics and Automation*, May 2015.

[41] G. Goretkin, A. Perez, Robert Platt, Jr., and G. Konidaris. Optimal sampling-based planning for linear-quadratic kinodynamic systems. In *IEEE International Conference on Robotics and Automation*, May 2013.

[42] R.L. Graham and Pavol Hell. On the history of the minimum spanning tree problem. *Annals of the History of Computing*, 7(1):43–57, Jan 1985.

[43] Jung-Su Ha, Ju-Jang Lee, and Han-Lim Choi. A successive approximation-based approach for optimal kinodynamic motion planning with nonlinear differential constraints. In *52nd IEEE Conference on Decision and Control*, December 2013.

[44] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, July 1968.

[45] Kris Hauser. Lazy collision checking in asymptotically-optimal motion planning. In *IEEE International Conference on Robotics and Automation*, May 2015.

[46] J.P.M. Hendrikx, T.J.J. Meijlink, and R.F.C. Kriens. Application of optimal control theory to inverse simulation of car handling. *Vehicle System Dynamics*, 26:449–461, 1996.

[47] Toshihiro Hiraoka, Osamu Nishihara, and Hiromitsu Kumamoto. Automatic path-tracking controller of a four-wheel steering vehicle. *Vehicle System Dynamics*, 47(10):1205–1227, 2009.

[48] Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *The International Journal of Robotics Research*, 2015.

[49] J. Jeon, R. V. Cowlagi, S. C. Peters, S. Karaman, E. Frazzoli, P. Tsiotras, and K. Iagnemma. Optimal motion planning with the half-car dynamical model for autonomous high-speed driving. In *Proceedings of the American Control Conference*, Washington, DC, June 2013.

[50] J. Jeon, S. Karaman, and E. Frazzoli. Optimal sampling-based feedback motion trees among obstacles for controllable linear systems with linear constraints. In *IEEE International Conference on Robotics and Automation*, Seattle, WA, June 2015.

[51] Jeong hwan Jeon, Sertac Karaman, and Emilio Frazzoli. Anytime computation of time-optimal off-road vehicle maneuvers using the RRT*. In *50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC)*, Orlando, FL, December 2011.

[52] In-Bae Jeong, Seung-Jae Lee, and Jong-Hwan Kim. RRT*-Quick: A motion planning algorithm with faster convergence rate. In *Robot Intelligence Technology and Applications 3*, volume 345 of *Advances in Intelligent Systems and Computing*, pages 67–76. Springer International Publishing, 2015.

[53] Thomas Kailath. *Linear Systems.* Prentice-Hall, New Jersey, 1980.

[54] S. Karaman and E. Frazzoli. Sampling-based optimal motion planning for non-holonomic dynamical systems. In *IEEE International Conference on Robotics and Automation*, Karlsruhe, Germany, May 2013.

[55] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller. Anytime motion planning using the RRT*. In *IEEE International Conference on Robotics and Automation*, Shanghai, China, May 2011.

[56] Sertac Karaman and Emilio Frazzoli. Optimal kinodynamic motion planning using incremental sampling-based methods. In *IEEE Conference on Decision and Control*, Atlanta, GA, December 2010.

[57] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, June 2011.

[58] Timur Karatas and Francesco Bullo. Randomized searches and nonlinear programming in trajectory planning. In *Proceedings of the 40th IEEE Conference on Decision and Control*, December 2001.

[59] Lydia E. Kavraki. Geometry and the discovery of new ligands. In *Workshop on the Algorithmic Foundations of Robotics*, 1996.

[60] Lydia E. Kavraki, Petr Švestka, Jean-Claude Latombe, and Mark H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, Aug 1996.

[61] Sven Koenig and Maxim Likhachev. D* Lite. In *Proceedings of the 18th National Conference on Artificial Intelligence*, 2002.

[62] Sven Koenig, Maxim Likhachev, and David Furcy. Lifelong planning A*. *Artificial Intelligence*, 155(1):93–146, 2004.

[63] Krisada Kritayakirana and J. Christian Gerdes. Using the centre of percussion to design a steering controller for an autonomous race car. *Vehicle System Dynamics*, 50(sup1):33–51, 2012.

[64] James Kuffner, Koichi Nishiwaki, Satoshi Kagami, Masayuki Inaba, and Hirochika Inoue. Motion planning for humanoid robots. In *Robotics Research. The Eleventh International Symposium*, pages 365–374, 2005.

[65] James J. Kuffner, Jr. and Steven M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation*, 2000.

[66] James J. Kuffner Jr, Satoshi Kagami, Koichi Nishiwaki, Masayuki Inaba, and Hirochika Inoue. Dynamically-stable motion planning for humanoid robots. *Autonomous Robots*, 12(1):105–118, 2002.

[67] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. P. How. Real-time motion planning with applications to autonomous urban driving. *Control Systems Technology, IEEE Transactions on*, 17(5):1105–1118, September 2009.

[68] Ailsa H. Land and Alison G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, Jul 1960.

[69] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[70] J.-P. Laumond, P. E. Jacobs, M. Taïx, and R. M. Murray. A motion planner for nonholonomic mobile robots. *Robotics and Automation, IEEE Transactions on*, 10:577–593, October 1994.

[71] J.P. Laumond, S. Sekhavat, and F. Lamiraux. Guidelines in nonholonomic motion planning for mobile robots. In *Robot Motion Planning and Control*, volume 229 of *Lecture Notes in Control and Information Sciences*, pages 1–53. Springer-Verlag, 1998.

[72] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[73] Steven M. LaValle, Michael S. Branicky, and Stephen R. Lindemann. On the relationship between classical grid search and probabilistic roadmaps. *The International Journal of Robotics Research*, 23(7–8):673–692, 2004.

[74] Steven M. LaValle and James J. Kuffner, Jr. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, May 2001.

[75] J. Leonard, J. How, S. Teller, M. Berger, S. Campbell, G. Fiore, L. Fletcher, E. Frazzoli, A. Huang, S. Karaman, O. Koch, Y. Kuwata, D. Moore, E. Olson, S. Peters, J. Teo, R. Truax, M. Walter, D. Barrett, A. Epstein, K. Maheloni, K. Moyer, T. Jones, R. Buckley, M. Antone, R. Galejs, S. Krishnamurthy, and J. Williams. A perception driven autonomous urban vehicle. *Journal of Field Robotics*, 25:727–774, 2008.

[76] Zakary Littlefield, Yanbo Li, and Kostas E. Bekris. Efficient sampling-based motion planning with asymptotic near-optimality guarantees for systems with dynamics. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, November 2013.

[77] David G Luenberger. Canonical forms for linear multivariable systems. *Automatic Control, IEEE Transactions on*, pages 290–293, June 1967.

[78] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *IEEE International Conference on Robotics and Automation*, May 2011.

[79] Mark B. Milam, Kudah Mushambi, and Richard M. Murray. A new computational approach to real-time trajectory generation for constrained mechanical systems. In *Proceedings of the 39th IEEE Conference on Decision and Control*, 2000.

[80] William F. Milliken and Douglas L. Milliken. *Race Car Vehicle Dynamics*. SAE International, 1995.

[81] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, and D. Haehnel. Junior: The Stanford entry in the Urban Challenge. *Journal of Field Robotics*, 25:569–597, 2008.

[82] Richard M. Murray and S. Shankar Sastry. Nonholonomic motion planning: Steering using sinusoids. *Automatic Control, IEEE Transactions on*, 38(5):700–716, 1993.

[83] J. Nasir, F. Islam, U. Malik, Y. Ayaz, O. Hasan, M. Khan, and M. S. Muhammad. RRT*-SMART: A rapid convergence implementation of RRT*. *International Journal of Advanced Robotic Systems*, 10, 2013.

[84] Reza Olfati-Saber. Near-identity diffeomorphisms and exponential ϵ-tracking and ϵ-stabilization of first-order nonholonomic se(2) vehicles. In *Proceedings of the American Control Conference*, volume 6, pages 4690–4695, 2002.

[85] Brian Paden, Sze Zheng Yong, Jeong hwan Jeon, and Emilio Frazzoli. A general formula for obtaining input and state trajectories from flat outputs of linear systems. In *IEEE Conference on Decision and Control*, December 2015. Submitted.

[86] Jun Peng, Wenhao Luo, Weirong Liu, Wentao Yu, and Jing Wang. A suboptimal and analytical solution to mobile robot trajectory generation amidst moving obstacles. *Autonomous Robots*, 2015.

[87] Mathew Penrose. *Random Geometric Graphs.* Oxford University Press Oxford, 2003.

[88] A. Perez, S. Karaman, A. Shkolnik, E. Frazzoli, S. Teller, and M. R. Walter. Asymptotically-optimal path planning for manipulation using incremental sampling-based algorithms. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, San Francisco, CA, September 2011.

[89] A. Perez, Robert Platt, Jr., G. Konidaris, L. Kaelbling, and T. Lozano-Perez. LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics. In *IEEE International Conference on Robotics and Automation*, May 2012.

[90] Sven Mikael Persson and Inna Sharf. Sampling-based A* algorithm for robot path-planning. *The International Journal of Robotics Research*, 33(13):1683–1708, 2014.

[91] Steven C. Peters, Emilio Frazzoli, and Karl Iagnemma. Differential flatness of a front-steered vehicle with tire force control. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, San Francisco, CA, September 2011.

[92] G. Petrone, C. Hill, and M. E. Biancolini. Track by track robust optimization of a F1 front wing using adjoint solutions and radial basis functions. In *44th AIAA Fluid Dynamics Conference*, 2014.

[93] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes: The Art of Scientific Computing.* Cambridge University Press, 2007.

[94] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, 1994.

[95] Zhihua Qu, Jing Wang, and Clinton E. Plaisted. A new analytical solution to mobile robot trajectory generation in the presence of moving obstacles. *Robotics, IEEE Transactions on*, 20(6):978–993, December 2004.

[96] Nathan Ratliff, Matt Zucker, J. Andrew Bagnell, and Siddhartha Srinivasa. CHOMP: Gradient optimization techniques for efficient motion planning. In *IEEE International Conference on Robotics and Automation*, pages 489–494, 2009.

[97] John H. Reif. Complexity of the mover's problem and generalizations. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, pages 421–427, 1979.

[98] Charles Richter, Adam Bry, and Nicholas Roy. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *Proceedings of the International Symposium of Robotics Research (ISRR 2013)*, December 2013.

[99] Gideon Sahar and John M. Hollerbach. Planning of minimum-time trajectories for robot arms. *The International Journal of Robotics Research*, 5(3):90–100, September 1986.

[100] Oren Salzman and Dan Halperin. Asymptotically near-optimal RRT for fast, high-quality, motion planning. In *IEEE International Conference on Robotics and Automation*, May 2014.

[101] Oren Salzman and Dan Halperin. Asymptotically-optimal motion planning using lower bounds on cost. In *IEEE International Conference on Robotics and Automation*, May 2015.

[102] Pradeep Setlur, John R. Wagner, Darren M. Dawson, and David Braganza. A trajectory tracking steer-by-wire control system for ground vehicles. *Vehicular Technology, IEEE Transactions on*, 55(1):76–85, 2006.

[103] Alexander Shkolnik and Russ Tedrake. Path planning in 1000+ dimensions using a task-space voronoi bias. In *IEEE International Conference on Robotics and Automation*, May 2009.

[104] Alexander Shkolnik, Matthew Walter, and Russ Tedrake. Reachability-guided sampling for planning under differential constraints. In *IEEE International Conference on Robotics and Automation*, May 2009.

[105] Thierry Siméon, Jean-Paul Laumond, Juan Cortés, and Anis Sahbani. Manipulation planning with probabilistic roadmaps. *The International Journal of Robotics Research*, 23(7-8):729–746, 2004.

[106] Anthony Stentz. The focussed D* algorithm for real-time replanning. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, 1995.

[107] Russell H. Taylor and Dan Stoianovici. Medical robotics in computer-integrated surgery. *Robotics and Automation, IEEE Transactions on*, 19(5):765–781, October 2003.

[108] Russ Tedrake, Ian R. Manchester, Mark Tobenkin, and John W. Roberts. LQR-trees: Feedback motion planning via sums-of-squares verification. *The International Journal of Robotics Research*, 29(8):1038–1052, July 2010.

[109] S. Teller, M. R. Walter, M. Antone, A. Correa, R. Davis, L. Fletcher, E. Frazzoli, J. Glass, J. P. How, A. S. Huang, J. Jeon, S. Karaman, B. Luders, N. Roy, and T. Sainath. A voice-commandable robotic forklift working alongside humans in

minimally-prepared outdoor environments. In *IEEE International Conference on Robotics and Automation*, Anchorage, AK, May 2010.

[110] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. N. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, M. Gittleman, S. Harbaugh, M. Hebert, T. M. Howard, S. Kolski, A. Kelly, M. Likhachev, M. McNaughton, N. Miller, K. Peterson, B. Pilnick, R. Rajkumar, P. Rybski, B. Salesky, Y. Seo, S. Singh, J. Snider, A. Stentz, W. R. Whittaker, Z. Wolkowicki, J. Ziglar, H. Bae, T. Brown, D. Demitrish, B. Litkouhi, J. Nickolaou, V. Sadekar, W. Zhang, J. Struble, M. Taylor, M. Darms, and D. Ferguson. Autonomous driving in urban environments: Boss and the Urban Challenge. *Journal of Field Robotics*, 25:425–466, 2008.

[111] Michael J. van Nieuwstadt and Richard M. Murray. Real-time trajectory generation for differentially flat systems. *International Journal of Robust and Nonlinear Control*, 8(11):995–1020, 1998.

[112] E. Velenis and P. Tsiotras. Optimal velocity profile generation for given acceleration limits: The half-car model case. In *Proceedings of the IEEE International Symposium on Industrial Electronics*, pages 361–366, Dubrovnik, Croatia, June 20–23 2005.

[113] 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*, 138(2):275–296, 2008.

[114] Efstathios Velenis, Emilio Frazzoli, and Panagiotis Tsiotras. Steady-state cornering equilibria and stabilization for a vehicle during extreme operating conditions. *Int. Journal of Vehicle Autonomous Systems*, 2010.

[115] Efstathios Velenis, Panagiotis Tsiotras, and Jianbo Lu. Optimality properties and driver input parameterization for trail-braking cornering. *European Journal of Control*, 4:308–320, 2008.

[116] M. R. Walter, M. Antone, E. Chuangsuwanich, A. Correa, R. Davis, L. Fletcher, E. Frazzoli, Y. Friedman, J. Glass, J. P. How, J. Jeon, S. Karaman, B. Luders, N. Roy, S. Tellex, and S. Teller. A situationally aware voice-commandable robotic forklift working alongside people in unstructured outdoor environments. *Journal of Field Robotics*, 32(4):590–628, 2015.

[117] Dustin J. Webb and Jur van den Berg. Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics. In *IEEE International Conference on Robotics and Automation*, May 2013.

[118] Christopher Xie, Jur van den Berg, Sachin Patil, and Pieter Abbeel. Toward asymptotically optimal motion planning for kinodynamic systems using a two-point boundary value problem solver. In *IEEE International Conference on Robotics and Automation*, May 2015.

[119] Jeffery H. Yakey, Steven M. LaValle, and Lydia E. Kavraki. Randomized path planning for linkages with closed kinematic chains. *Robotics and Automation, IEEE Transactions on*, 17(6):951–958, Dec 2001.

[120] Holly A Yanco, Adam Norton, Willard Ober, David Shane, Anna Skinner, and Jack Vice. Analysis of human-robot interaction at the DARPA Robotics Challenge Trials. *Journal of Field Robotics*, 32(3):420–444, 2015.

[121] Dmitry S. Yershov and Emilio Frazzoli. Asymptotically optimal feedback planning: FMM meets adaptive mesh refinement. In *Workshop on the Algorithmic Foundations of Robotics*, 2014.

[122] Dmitry S. Yershov and Steven M. LaValle. Simplicial Dijkstra and A^* algorithms for optimal feedback planning. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, September 2011.

[123] Shlomo Zilberstein. Operational rationality through compilation of anytime algorithms. *AI Magazine*, 16(2):79–80, 1995.

[124] Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996.