Reducing RRT metric sensitivity for motion planning with differential constraints

by

Peng Cheng

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Major Professor: Steven M. LaValle

Iowa State University

Ames, Iowa

Graduate College
Iowa State University

This is to certify that the Master's thesis of

Peng Cheng

has met the thesis requirements of Iowa State University

Major Professor

For the Major Program

For the Graduate College

To my beloved wife, parents, and sisters

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The Rapidly-Exploring Random Tree (RRT) is a search structure constructed incrementally in a way that quickly reduces the expected distance to nodes in the search structure, of randomly-chosen points in the search space. Although it has recently shown promising results in trajectory design problems, its performance is not reliable for very challenging problems with differential constraints. One reason is metric sensitivity, which measures the degrading of the performance of the planner due to poor guidance information provided by the metric function. It is typical for robotics problems with high-dimensional, nonlinear systems, in which insufficient problem domain knowledge can be provided to construct a good metric function.

The thesis presents improved methods to reduce the negative effect of a poor metric function. Two main ideas are employed: 1) Combining systematic search with a randomized algorithm. 2) Collecting the constraint violation frequency during the exploration. The constraint violation frequency gathers global constraint information and helps planners to make better choices in the search. The derived planners possess both the completeness from systematic search and fast exploration properties from the RRT. Our analysis shows that the new methods: 1) are resolution complete, 2) are more reliable than the original RRT, and 3) have a deterministic upper bound on the required space and iterations. The resolution completeness theorem is also applicable to other discrete time and discretized input space planners. We have implemented the algorithm and demonstrated that it can solve challenging trajectory design problems involving a 9-dimensional nonlinear car-like robot and a 12-dimensional underactuated spacecraft.

CHAPTER 1 INTRODUCTION

Robotics is the science or study of the technology associated with the theory, design, fabrication, and application of robots to autonomously complete tedious or dangerous work, such as housework, deep sea mining, fires extinguishing, and outer space exploration. Designing such robots is a formidable task because it incorporates problems from different disciplines, such as system engineering, material engineering, mechanical engineering, computer science, and so on. The motion planning problem addresses how to automatically drive a robot from one state to another state safely. It is critical to robotics because robots will usually have to solve this problem during the execution of the assigned job. It is also a difficult problem when you think in the perspective of a robot: "How I can find a path automatically in the various kinds of different complex environments?".

1.1 Challenges in Motion Planning

Motion planning problem is hard even for a simple Piano Movers' problem [80], in which a robot is a polyhedral, obstacles consist of polyhedral walls, and the objective is to find a continuous collision-free path connecting the starting and goal configurations. Planning a collision-free path between any two configurations for a robot made of an arbitrary number of polyhedral bodies connected together at some joint vertices, among a finite set of polyhedral obstacles, was shown to be a PSPACE-hard problem by Reif [75]. Canny and Reif [16] proved that a compliant motion problem for a point robot in the presence of uncertainty, in a three-dimensional polyhedral configuration space with an arbitrarily large number of faces, is a nondeterministic exponential time hard (NEXPTIME-hard) problem. With moving obstacles in a three-dimensional workspace and a robot with bounded velocity, planning problems were shown to be PSPACE-hard by Reif and Sharir [76]. They also showed that for robots with unbounded velocity, the problem is NP-hard.

Recently motion planning problems have become more complicated from the initial simple problems only considering geometric constraints [30, 80, 82, 81, 83, 84] to problems considering differential constraints, in which differential geometric constraints lead to nonholonomic path planning problems [54, 55] and dynamic constraints generate kinodynamic path planning problems [25, 72, 88]. To formulate these more challenging problems, the search space is changed from the configuration space [95], the set of all geometric transformations, to the state space [25] to include the time derivatives of configuration variables. A direct impact of this variation is that the dimension of the state space is normally at least twice of that of the configuration space. Another effect is that purely geometric methods developed in classical path planning cannot be applied directly to ones with differential constraints. Even in the absence of obstacles, there is no general nonholonomic planning method which is guaranteed to find an exact solution for any given goal state. Existing deterministic approximate algorithms [4] find solution with a given tolerance, but have limited application because of the curse of dimensionality. With the increase of the dimension of the problem space, most of the classical path planning methods, such as skeleton methods [13, 71, 16, 17], cell decomposition [14, 30, 64, 80], potential field [44, 46, 37], and so on, become impractical. The inapplicability of these methods is that these methods either are based on the systematic search or use the heuristic function with the local minimum. Some navigation functions based methods have no local minimum [47, 22, 2, 77, 78, 90], but constructing them for high-dimensional problems is time consuming. Recently, the application of randomized techniques in motion planning generates many successful methods for the high-dimensional problems, such as Rapidly-Exploring Random Trees (RRTs) [60, 61], randomized potential fields [3, 18, 44], probabilistic roadmaps [1, 43], Ariadne's clew algorithm [68, 67], and the planners in [35, 99]. Although randomized methods can solve many high-dimensional problems, there are still many shortcomings, such as parameter tuning, metric sensitivity, lack of performance analysis, and so on. Solutions to these problems will lead to a better understanding of randomized algorithms and can make randomized algorithms to be more reliable and more efficient.

1.2 Challenges in Trajectory Design

In control literature, considerable research has also been done on motion planning problems; however, a solution is a control function and no obstacles in the environment are considered. Control problems are related to path planning problems that consider differential constraints, such as kinodynamic path planning [15, 20, 21, 23, 24, 25, 32, 58, 66, 74, 79, 85], in which the dynamics of the robot system is considered. In control theory, a solution normally takes the form of a closed analytically expressed control law. For example, the point-to-point minimum energy control law for a linear system can be computed in closed form by solving a matrix Lyapunov equation [19]. When an analytical solution is not available, numerical techniques can be employed. Although there is much recent progress in nonlinear control, it still seems difficult to solve problems that consider both a nonlinear system equation and non-convex constraints on the state space. The attempt to solve these problems by using a numerical algorithm based on dynamic programming produces limited success on low-dimensional problems [5, 8, 50, 51]. Similar to the motion planning problem, applying randomized techniques in the trajectory design is a promising approach.

1.3 Content of the Thesis

Inspired by the idea of Rapidly-Exploring Random Trees (RRTs), which are constructed incrementally in a way that quickly reduces the expected distance to the tree of randomly-chosen points in the state space, the methods in this thesis combine concepts and methods from randomized algorithms, motion planning, artificial intelligence and mathematical programming to improve RRTs by reducing the metric sensitivity of the original RRT through collecting and utilizing the information during the search process. The performance of the new method will not decay greatly even if a poor metric heuristic function is constructed when there is insufficient problem domain knowledge. The improvement is verified by comparing computed results between the new method and original RRT in trajectory design experiments. The resolution completeness of the algorithm, comparison of the new method with the original RRT, and worst-case analysis are provided.

Chapter 2 provides a detailed literature review and the motivation for this thesis. The problems considered in this thesis are formulated in Chapter 3. Several examples are also presented, which are used in experiments in Chapter 6. New methods are presented in Chapter 4 and their analysis is in Chapter 5. In Chapter 6, performance comparison of the improved RRT and the original RRT is done for lane changing experiments. Two challenging experiments, virtual driving with 9-dimensional car model and trajectory design for 12-dimensional spacecraft, are solved by the improved RRT-based planners. Conclusions are given in Chapter 7.

CHAPTER 2 RELATED RESEARCH AND MOTIVATIONS

Much research has been done in the field of motion planning and trajectory design. Only relevant approaches of interest to the applications in this thesis will be described.

2.1 Current Motion Planning Interest

Motion planning research can be divided into two phases: Phase one includes motion planning problems which ignore differential constraints. The configuration space concept [95], which was borrowed from mechanics [65], was used as the problem formulation tool. Every geometric transformation that can be applied to the robot is reduced to a point in the configuration space. Many different planning methods [30, 82, 80, 81, 83, 84] are developed from this tool. A survey of motion planning techniques for holonomic robots appears in [36].

Phase two began after the characterization of the first nonholonomic path planning problem by Laumond [54], in which systems are characterized by the non-integrable constraints among the derivatives of system configuration variables. Differential constraints exist generally in nature. They appear in systems in form of either differential geometric constraints or dynamics constraints. Kinematic models considering non-integrable differential geometric constraints are common for nonholonomic path planning problems, such as the car-like robot with or without trailers [4], dextrous manipulation when the spherical tips of the robot's finger perform rolling motions when in contact with an object [63], or a robotic manipulator that pushes objects from one place to another [66].

To apply the solution path to real robots in practice, a trajectory or an input sequence, instead of the path, is needed. The trajectory or the input sequence is mostly generated from the path, which is usually generated from path planning methods ignoring differential constraints. However, for robots with high dynamic constraints, such as high-fidelity models of air, land, and underwater vehicles, a path that is found using kinematic path planning, might be unexecutable. Thus, dynamic differential constraints are considered in motion planning problems, which leads to the emergence of kinodynamic path planning problems [25, 23].

2.1.1 Nonholonomic motion planning

The consideration of accessible configurations for robots with nonholonomic constraints led to the introduction of the controllability concept [33] from the system control into robotics literature [57, 62]. If the set of configurations reachable from any configuration q contains a neighborhood of q, then the system is *locally controllable*. It is *small-time controllable* if the set of configurations, that are reachable from q before any given strictly positive time T, contains a neighborhood of q. Small-time controllability implies local controllability, but the converse is false. Linear equality differential constraints are considered in [57], in which if the controllability rank condition, the dimension of the Lie algebra of the vector fields generated by the controls for the system equals to the dimension of the configuration space, is satisfied, the system is small-time controllable. In [62], Li and Canny showed that a ball can reach any configuration on a plane by a pure rolling motion. A more general result considering both equality and inequality differential constraints was presented in [4], in which a multi-body mobile robot is controllable if the steering angle φ of the car can take at least two different values φ_1 and φ_2 in $(-\pi, +\pi]$ such that $|\varphi_2 - \varphi_1| \neq \pi$. A detailed survey of the research in nonholonomic motion planning is in [56].

Under differential constraints, even without obstacles, there are only exact solution methods for special classes of systems. No existing algorithms can be applied on general systems.

Steering methods are used for motion planning problems with nonholonomic constraints in obstacle-free environments. Dubin's [26] found the shortest path considering no obstacles for a car moving forward with a constant velocity, in which the shortest paths to linking any two configurations consist of a sequence of curves which are chosen from a set of six curves. For a car that can move both forward and backward and has a minimal turning radius, Reeds and Shepp [73] proved that the shortest path can always be chosen among 48 simple paths constituted by at most five pieces which are straight line segments or arcs of circle with minimal radius. Sussmann and Tang [91] reduced the sufficient family to 46 different kinds of paths. Souéres and Laumond [89] showed a way to select inside this family an optimal path to link any two configurations. For more difficult problems such as cars that pull trailers, steering methods have been proposed which provide a non-optimal control function which drives the robot to its goal configuration [48, 69].

Considering a point robot in an environment with polygonal obstacles, a complete algorithm for deciding the existence of a feasible path between any two configurations was presented by Fortune and Wilfong [28]. In [38], an approximate solution to the above problem was given. For a workspace that contains only "moderated" obstacles, which are generalized polygons whose boundaries are accessible for Dubin's robot, an exact solution has been proposed in [11].

For a rigid car-like robot in an environment with polygonal obstacles, a complete algorithm for deciding the existence of a feasible path exists. Laumond [54, 55] designed a two-phase exact planner which is based on controllability of a single-body mobile robot that uses the two basic maneuvers. In the first phase, a path is generated by holonomic path planning. In the second phase, the path is transformed into a feasible trajectory by combining the above two maneuvers. Using similar ideas based on the shortest path for the Reeds-Shepp car-like robot in [73], three-step planners were developed for car-like robots in [39, 93, 52]. Besides the first two same steps as in [54, 55], the third step tries to optimize the feasible path from step two. However, the resulting solution is not optimal because step two and step three are local procedures.

Most of the planners discussed above can only be applied to a specific class of systems. Barraquand and Latombe [4] described an approximate algorithm which is applicable to a wide range of problems. Based on the discretized configuration space, the method generates incrementally a graph starting from the initial state. In each iteration, a state that has the least cost in the graph is chosen. Successors are generated by integrating the state with different inputs for a fixed time. If the successors are in the cells which are not occupied by other states in the graph, they will be added to the graph. When the goal state shares the same cell with some state in the graph, a solution is reported. The algorithm approximates optimal paths and is proved asymptotically complete with respect to the fixed time and size of the cell for car-like robots. That is, for any given problem that admits a solution path, the planner is guaranteed to generate a solution path, provided that the discretization and fixed time have been set fine enough. Because it is based on dynamic programming, it is limited to lower dimensional problems. The proof of the asymptotic completeness is limited to the car-like robot only.

2.1.2 Kinodynamic path planning and trajectory design

Kinematic constraints restrict the collision-free configuration space, including joint limits and obstacles. Dynamic constraints restrict higher order time derivatives of the configuration, including dynamics laws and bounds on velocity, acceleration, and applied force. A general kinodynamic planning problem [25] (called trajectory design in control literature) is to find a path for a robot from the initial state to the goal state without violating differential constraints, in which the state represents both the configuration and velocity of the robot, and the differential constraints consist of both kinematic and dynamic constraints. Kinodynamic path planning is a difficult problem. In [24], the optimal Cartesian kinodynamic planning in 3D is shown to be NP-hard by extending Canny's and Reif's proof [16] of the NP-hardness of the 3D shortest-path problem.

The classic trajectory design approach in robotics research has been to decouple the general robotics problem by solving basic holonomic path planning, which takes into account obstacles while ignoring differential constraints, and then find a trajectory and controller that satisfies the differential constraints and tracks the path [10, 53, 86]. However, the result of a holonomic planner might be unexecutable by the robot due to dynamic constraints. Approaches that avoid decoupling have been proposed recently. In [70], ÓDúnlaing provides an exact polynomial-time algorithm for one dimensional kinodynamic planning, in which a point moving on the real line such that, given two functions, $f(t)$ and $g(t)$, the position of the point, $x(t)$, satisfies $\forall t, f(t) \leq x(t) \leq g(t)$. In [10, 87], it is shown that the velocity profile of a given trajectory can be rescaled so as to respect dynamic constraints and to be time-optimal. Based on these ideas, approximate methods computing near optimal trajectories for robots with several degrees of freedom and full dynamics by using a fixed-resolution configuration space or state space grid are developed in [79, 85], but these methods do not bound the goodness of the approximation. In [24, 23], a similar method based on the grid built from the acceleration bound and fixed time step was presented with bound on the goodness of the approximate optimal path and running time. It can find a safe trajectory with $1+\varepsilon$ times the time for an optimal trajectory, in which the safe trajectory means there exists a tube of a given size around the trajectory and ε is the approximation tolerance. All of these methods are derived from classical dynamic programming ideas which have been employed to yield numerical optimal control solutions in [5, 8, 50, 51]. Other methods using this idea in various forms in the motion planning and robotics literature are in [15, 20, 21, 32, 58, 66, 74].

In the control field, when no analytical structure can be exploited, trajectory optimization is tackled with numerical tools. Because this problem is infinitely dimensional, various forms of transcription (discretization) is usually employed to transform the original variational problem into a nonlinear program. Established transcription techniques include single, multiple shooting, or collocation techniques. Recent research results suggest that in fact, direct methods based on collocation are preferable. An early reference is [31], and [9, 27, 96] are recent surveys. Although the above numerical techniques have many successful applications, they are usually limited in practice to the problems with obstacle-free environments. The trend to combine the global techniques in motion planning and local planners based on the collocation and nonlinear programming to solve the nonlinear dynamic systems involving environments with complex obstacles appears recently and shows encouraging results [41]. In [29], a method combined a randomized global search (RRT) with traditional control to design the trajectory for autonomous vehicles moving in the environment with obstacles. The above method increases the planner performance from a control perspective by using nonlinear programming techniques to improve the local planner. Our method in this thesis is to improve the planner from an algorithmic perspective to make the global techniques more robust and efficient.

2.2 Randomized Path Planning

One of the main obstacles in the advancing of the motion planning is high dimensional problems because to find an optimal solution and keep completeness, dynamic programming techniques seems to be always required. Due to the curse of dimensionality, dynamic programming methods are impractical for the generic, high-dimensional problems considered in this thesis.

Attempts to fight the curse of dimensionality have led to the introduction of randomized (or Monte-Carlo) approaches into motion planning and trajectory design. The randomized path planner (RPP) [3] is the first randomized planner in motion planning literature, which used random walks to escape the local minima in the potential function and solved problems involving robots with 3 to 31 dof. Its analysis, based on Markov chains, was proposed in [49]. Because pure random walks in the algorithm do not use any heuristics, it is difficult for this method to ensure reliable performance. Probabilistic Roadmap planners (PRMs) [12, 34, 43, 92] construct an undirected graph by connecting using simple canonical paths (typically, straight-line segments in the configuration space) between neighboring collision-free configurations which are chosen randomly. Its performance analysis can be found in [42]. Due to the probabilistic complete covering of the configuration space, the method has reliable performance; however, the PRM is difficult to deal with problems with differential constraints. Random Neighborhood Graph (RNG) [97] presents a randomized framework for computing feedback motion strategy by defining a global navigation function over a collection of spherical balls in the configuration space. It features a Bayesian termination condition to guarantee any given success rate and currently solves up to six dimensional problems. Despite the success of these randomized techniques, they are limited in the holonomic path planning problems because their local planner normally uses a straight line to connect the two states, which is not suitable for the problem with differential constraints. Algorithms based on Rapidly-exploring Random Trees (RRTs) [59, 60] have been proposed recently for trajectory design problems that involve dynamics and complicated obstacle constraints. An RRT achieves rapid exploration by iteratively sampling a random state in the state space and extending the nearest state in the RRT to get as close as possible to the random state. Various RRT-based planners have been designed recently for autonomous vehicle motion planning [29] and for nonlinear underactuated vehicles [94]. A trajectory planner also based on randomized incremental search was proposed for time-varying systems in [45].

In spite of the successes of RRTs, one of the key shortcomings is the sensitivity of their performance with respect to a chosen metric on the state space. The metric serves as a guide to improve performance; however, for systems that involve substantial momentum, the metric can provide misleading information that dramatically increases the computation time. For some systems, it may be possible to design better metrics (as in the hybrid optimal cost-to-go function in [29]), but in general there is a need to develop randomized trajectory design algorithms that achieve reliable performance in spite of a poor metric. It is this demand that leads to the emergence of methods in this thesis.

2.3 Motivation of the Thesis

Our methods acquire inspirations from the above research in motion planning, trajectory design, mathematical programming, and artificial intelligence. The motivation can be classified into the following three aspects:

2.3.1 Combining systematic search and randomized techniques

Dynamic programming methods are complete and can find an approximate optimal solution if one exists. However, because it is a systematic search, it is impractical for high-dimensional problems. Randomized techniques applied in motion planning methods sacrifice the completeness and optimality for the efficiency. Instead of deterministic completeness, a probabilistic completeness is applied in the current randomized motion planning methods. A method which combines the systematic search and random techniques might be good for high-dimensional path planning problems. It may have the completeness of the systematic search and also have the same efficient exploration ability as the randomized planner.

2.3.2 Searching for a trajectory incrementally

Many holonomic path planners are unsuitable for trajectory design problems with differential constraints. The main reason is that these methods solve motion planning problems in two steps. Even though a solution from a holonomic planner exists in the first step, it might be unexecutable because of the differential constraints. For example, some states on the trajectory, that is generated from the path, might be unaccessible because they are in the collision region, or the trajectory cannot be followed because of the dynamic constraints. However, if we search for the trajectory directly, the resulting solution will automatically meet the differential constraints. To enable the search, inputs have to be in a finite discrete set and the configuration space or the state space should be bounded; fortunately, for computer-controlled robot systems, these conditions are always satisfied.

In [8], Bertsekas proved the convergence of discretization procedures in the dynamic programming. In his proof, both discretization in the state space and the control space are considered; the approximate solution will converge to the exact solution with sufficient resolution. Even though he proved the convergence of the numerical dynamic programming, his idea is basic for many other search methods.

2.3.3 Backtracking to avoid traps

This idea comes from the classical AI techniques. Imagine in a game of chess, you get into a check-mate state and have a chance to recover from the check-mate. Backtracking one step from the check-mate state is usually insufficient to avoid the check-mate. How many steps should be backtracked? For a master, he will consider all of the possible following movements in his mind and backtrack to a state which will have the least possibility to get the check-mate. A naive player might stick to backtracking one step and choose movements one by one. Only when all of the possible movements in backtracking one step are tested, he will backtrack two steps. For a moderate player who cannot consider all of the possible movements, the following strategy might be better than that of the naive player: The player will remember the effect of his attempts. When he finds that several movements fail after backtracking one step, he will backtrack several steps and is more likely to find a state to avoid the check-mate. A similar situation exists in motion planning. If a path runs into obstacles, new branches will be derived from the nodes along the collision path to extend to the goal state. The closer the node is to the collision state, the collision possibility of the new branch is greater. Instead of the master player, which considers all the possible paths and is impractical for motion planning, and the naive player, which is not efficient, the moderate player's strategy is employed in this thesis. The constraint violation frequency information, which corresponds to the goodness of the current state, will be collected only from the explored space and be used to help the planner to make better backtracking.

Unifying the above motivations together, our methods achieve the objective by inheriting randomized techniques and trajectory search from the RRT-based planner, incorporating the systematic search by recording exploration information about the explored space, and utilizing the backtracking strategy by collecting constraint violation frequency information.

CHAPTER 3 PROBLEM DESCRIPTION

In this chapter, we will first present the mathematical formulation of problems considered in this thesis. In Section 3.2, several typical problems will be introduced.

3.1 Mathematical Formulation

Trajectory design problems considered in this thesis are defined as $(X, D, x_{init}, X_{goal}, U, \delta t, \rho, \tau, f)$, in which the components are formulated as follows:

1. X**, state space:** An n-dimensional closed and bounded differentiable manifold, $X = \{x = (x_1, x_2, \cdots, x_n)\} \subset R^n$.

2. D**, constraint satisfaction function:** A function, $D : X \rightarrow \{true, false\}$, that determines whether global constraints are satisfied for a state x, in which $true$ means that global constraints are satisfied and $false$ means that global constraints are not satisfied. The global constraints are in form of $g(x) > 0$. Combining X and D, we define $X_{free} = \{x \mid x \in X, D(x) = true\} = \{x \mid x \in X, g(x) > 0\}$, which denotes the states in X that satisfy the constraints.

3. x_{init}, X_{goal}**, boundary conditions:** $x_{init} \in X_{free}$ is the initial state and $X_{goal} \subset X_{free}$ is the set of goal states.

4. U**, finite input set:** The complete set of m_u-dimensional inputs that can affect the state; U is independent of the state. For a continuous input set, some procedures

can be used to discretize the continuous input. The $u_i \in [u_{imin}, u_{imax}], i = 1, \cdots, m$ represents an input in U that has m inputs.

5. δt**, fixed control period:** A strictly positive real number which denotes the period during which $u \in U$ is applied.

6. ρ**, metric function:** A function $\rho : X \times X \rightarrow [0, \infty)$ defined on X.

7. τ**, solution tolerance:** A positive real number, τ, which denotes the precision requirement for the solution. For an exact algorithm, the end of a trajectory needs to lie in X_{goal} such that $\tau = 0$; for an approximate algorithm, it must within τ of X_{goal}.

8. f**, equation of motion:** An equation, $\dot{x} = f(x, u, t)$, that characterizes the evolution of the state and represents differential constraints of the systems. An incremental simulator integrates this equation to obtain future states.

The objective of the trajectory design problem is to find a piecewise-constant *solution control function*, $u : [t_0, t_f] \rightarrow U$, in which t_0 is the starting time and $t_f = t_0 + K\delta t$, and its corresponding *solution path,* $\pi : [t_0, t_f] \rightarrow X_{free}$, which meets the following conditions:

1. $u(t) = u_i \in U, \forall t \in [t_0 + k\delta t, t_0 + (k+1)\delta t), i = 1, 2, \cdots, m, k = 0, 1, \cdots, K-1.$

2. $\pi(t) \in X_{free}, t \in [t_0, t_f]$ in which,

$$\pi(t) = \begin{cases} x_{init} & \text{if } t = t_0 \\ \pi(t_0 + k\delta t) + \int_{t_0+k\delta t}^{t} f(x, u, t)dt & \text{if } t \in [t_0 + k\delta t, t_0 + (k+1)\delta t), \\ & \text{in which } k = 0, 1, \cdots, K-1, \end{cases} \tag{3.1}$$

3. $\exists\, x_{goal} \in X_{goal},\ \rho(\pi(t_f), x_{goal}) \leq \tau$.

3.2 Trajectory Design Problems

In this section, we introduce several typical trajectory design problems, which will be used in the experiments to verify the performance of algorithms.

3.2.1 Virtual driving

This section applies the trajectory design problem formulation to virtual prototyping. Suppose, for example, that one would like to evaluate proposed designs for a sports utility vehicle. A trajectory design algorithm in cooperation with a high-quality simulator could serve as a virtual "stunt driver" that evaluates the vehicle. One could ask questions such as "Can this SUV perform a rapid lane change without losing control?" or "Is it possible that the SUV could flip over sideways through some combination of inputs?". Similar problems exist in the prototyping of aircraft, spacecraft, hovercrafts, submarines, and a wide variety of mechanical machinery.

In this thesis, a 9-dimensional car model adapted from [7] is used to simulate the car. Our planner will drive the virtual car through a virtual town at high speed. To define the trajectory design problem, we first introduce the terms which will be used in the problem formulation. Fig. 3.1 and Fig. 3.2 show the parameters.

1. **World 3D coordinate frame, GC:** It is a global frame with the x coordinate increasing from left to right, the y coordinate increasing from top to bottom, and the z coordinate inward (to form a right-handed coordinate system). Rotation around the axis of the frame is also according to the right-hand rule. For example, the positive direction of the rotation around z axis is from the x axis to the y axis.

2. **3D coordinate frame fixed on car, LC:** It is a local frame fixed on the car,

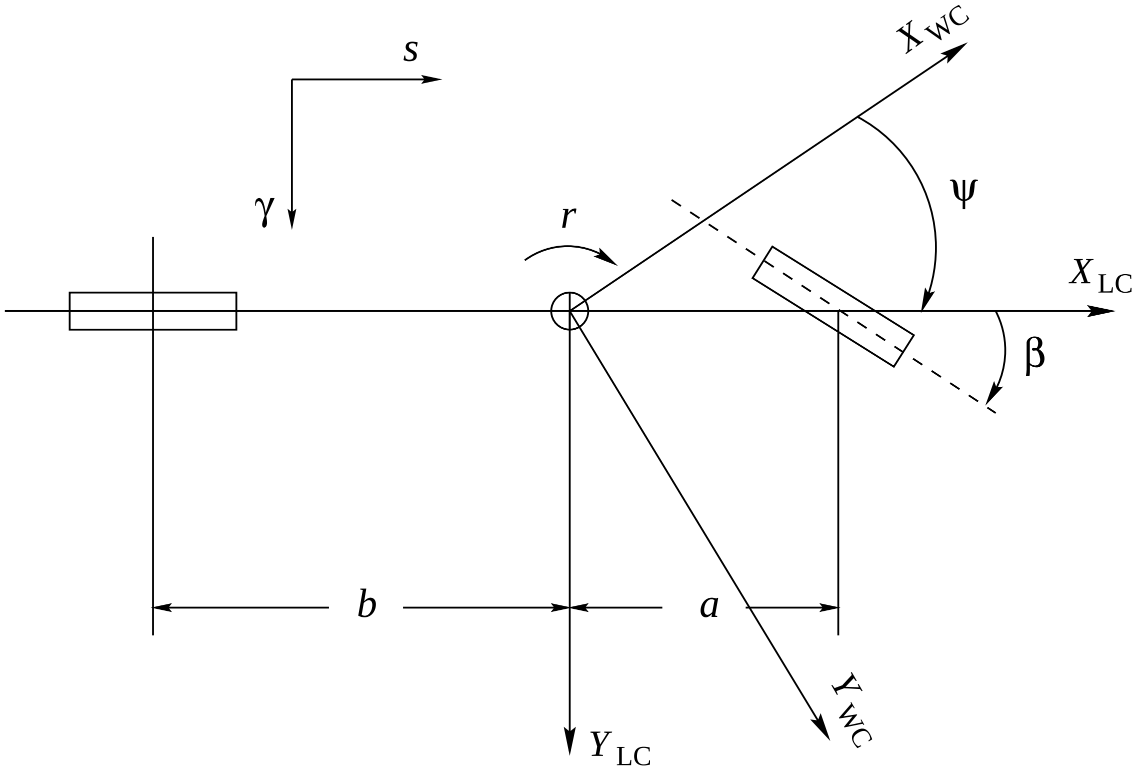


Figure 3.1 Top view of the car model.

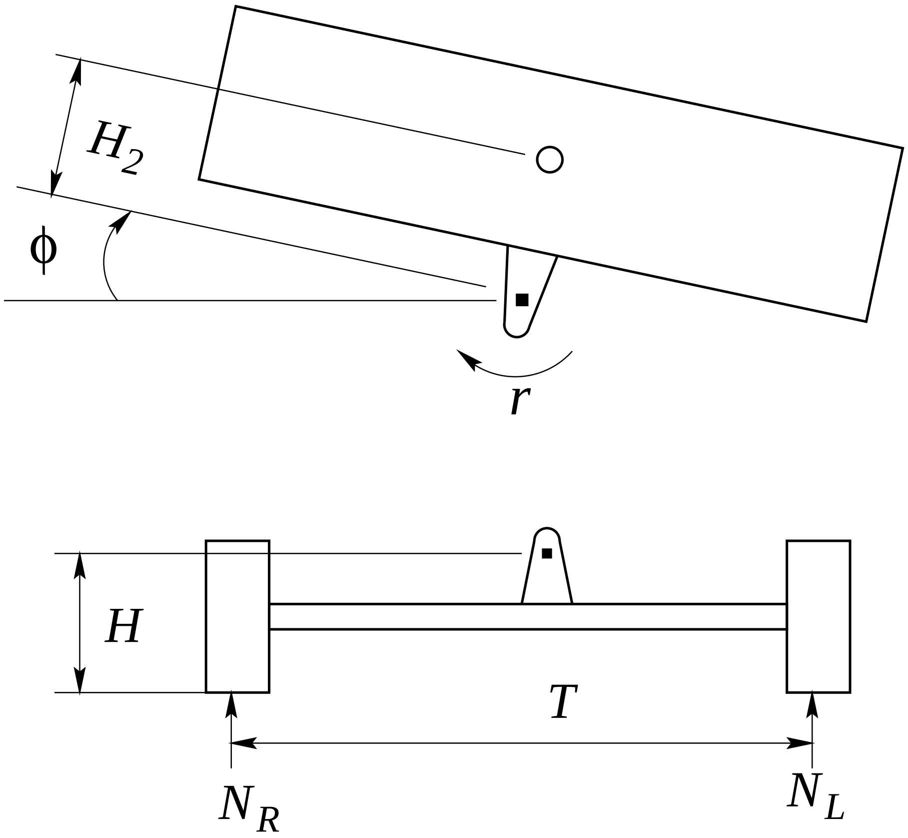


Figure 3.2 Front view of the car model.

with x extending from the mass center to the center of the front axle, z extending inward, and y determined according to the right-handed rule.

3. ψ**:** The yaw angle of the car that corresponds to the rotation around z of GC.

4. r**:** The velocity of yaw angle of the car.

5. ϕ**:** The roll angle of the car that describes the rotation around x of LC.

6. q**:** The velocity of the roll angle.

7. ν**:** The sideways speed (arising from slipping) that is with respect to the y in LC.

8. β**:** The steering angle is with respect to z of LC.

9. s**:** The forward speed of the car that is with respect to x of LC.

10. $C_{\alpha f}, C_{\alpha r}$**:** They are the total cornering stiffness of the front and rear wheels, respectively.

11. M**:** The car mass

12. I_y**:** The yaw moment of inertia of the car.

13. I_r**:** The roll moment of inertia of the car.

14. a, b**:** The distance from the center of the front and rear axles to the car mass center, respectively, with respect to LC.

15. H**:** The height of the joint connecting the chassis with the car frame (the chassis and frame are flexibly attached to model a simple suspension system).

16. T**:** The distance between the left and right wheels.

17. H_2**:** The distance from the joint to the mass center of the car frame.

18. K, c, μ**:** Constant parameters which are described in [7].

Based on these definitions, the corresponding formulation of the virtual driving problem is given as follows:

1. **9-Dimensional State Space:** A state x in X is represented by $(x, y, r, \psi, \phi, q, \nu, s, \beta)$.

2. **2-Dimensional Input Set:** $u = (u_1, u_2)$, in which u_1 is linear acceleration and u_2 is the rate of the change of the steering angle. To get a finite input set, we can discretize the region between $[u_{imin}, u_{imax}]$ for $i = 1, 2$, respectively, and combine the discretized value in two regions to get all the possible inputs.

3. **Equation of motion:** Let α_f and α_r be the slipping angle of the front and rear wheels, respectively, with respect to z of LC. They are expressed as

$$\alpha_f = \frac{\nu + ar}{s} - \beta$$

and

$$\alpha_r = \frac{\nu - br}{s}.$$

Because of the rolling effect, the load on the wheels are different. Let $N_{fl}, N_{fr}, N_{rl}, N_{rr}$ be the load, respectively, on the front left, front right, rear left, and rear right wheels, and $F_{yfl}, F_{yfr}, F_{yrl}, F_{yrr}$ be the force along y in LC, respectively, on the front left, front right, rear left, and rear right wheels. Considering under some conditions it is possible for the car to slip sideways. The y-direction forces on four wheels are:

$$F_{yfl} = -C_{\alpha f}\alpha_f/2, \ \ \mathit{if}\ N_{fl}\mu > C_{\alpha f}\tan(|\alpha_f|)/2$$

$$F_{yfr} = -C_{\alpha f}\alpha_f/2, \ \ \mathit{if}\ N_{fr}\mu > C_{\alpha f}\tan(|\alpha_f|)/2$$

$$F_{yrl} = -C_{\alpha r}\alpha_r/2,\ \ if\ N_{rl}\mu > C_{\alpha r}\tan(|\alpha_r|)/2$$

$$F_{yrr} = -C_{\alpha r}\alpha_r/2,\ \ if\ N_{rr}\mu/2 > C_{\alpha r}\tan(|\alpha_r|)/2\ ,$$

in which the calculated friction force is less than the maximum possible friction. Otherwise,

$$F_{yfl} = -\mu N_{fl}\mathrm{Sgn}(\alpha_f)(1 - x_{fl}/2), x_{fl} = N_{fl}\mu C_{\alpha f}\tan(|\alpha_f|)/2$$

$$F_{yfr} = -\mu N_{fr}\mathrm{Sgn}(\alpha_f)(1 - x_{fr}/2), x_{fr} = N_{fr}\mu C_{\alpha f}\tan(|\alpha_f|)/2$$

$$F_{yrl} = -\mu N_{rl}\mathrm{Sgn}(\alpha_r)(1 - x_{rl}/2), x_{rl} = N_{fl}\mu C_{\alpha r}\tan(|\alpha_r|)/2$$

$$F_{yrr} = -\mu N_{rr}\mathrm{Sgn}(\alpha_r)(1 - x_{rr}/2), x_{rr} = N_{fr}\mu C_{\alpha r}\tan(|\alpha_r|)/2.$$

Let $F_{yf} = F_{yfl}+F_{yfr}$, $F_{yr} = F_{yrl}+F_{yrr}$, $h = (-(K-MgH_2)\phi-cq-(F_{yf}+F_{yr})H_2)/I_r$, and $\lambda = \frac{b}{a+b}$. The difference between the left wheels and right wheels is: $N_{dif} = \frac{(-K\phi-Cq+(F_{yf}+F_{yr})H)}{T/2}$, and current load on different wheels is changed to be:

$$N_{fl} = 9.8M\lambda/2 + N_{dif}\lambda/2$$

$$N_{fr} = 9.8M\lambda/2 - N_{dif}\lambda/2$$

$$N_{rl} = 9.8M(1-\lambda)/2 + N_{dif}(1-\lambda)/2$$

$$N_{rr} = 9.8M(1-\lambda)/2 - N_{dif}(1-\lambda)/2.$$

To keep the car from rolling over, load on every wheel should greater than zero. If one of the loads is less than zero, the car is in a dangerous condition. The nine equations of motion are:

$$\dot{x} = s\cos\psi - \nu\sin\psi,$$

$$\dot{y} = s\sin\psi + \nu\cos\psi,$$

$$\dot{r} = (F_{yf}a - F_{yr}b)/I_y,$$

$$\dot{q} = h,\ \dot{\psi} = r,\ \dot{\phi} = q,$$

$$\dot{\nu} = (F_{yf} + F_{yr})/M - sr - H_2 h,$$

$$\dot{s} = u_1,\ \dot{\beta} = u_2.$$

If the rolling effect is ignored and the forward speed is fixed, a 5-dimensional nonlinear car model can be derived from the 9-dimensional car model. This model will be used in lane changing experiments, in which a car is required to change lane at high speed, to comparing the performance of new methods and the original RRT-based methods. The simplified car model is:

1. **5-Dimensional state space:** A state x in X is represented by (x, y, r, ψ, ν).

2. **1-Dimensional input set:** The input vector $u = (u_1)$ is the steering angle. The input set represents the discretized steering angles.

3. **Equation of motion:**

   The slipping angle of the front and rear wheels are:

   $$\alpha_f = \frac{\nu + ar}{s} - u_1$$

   and

   $$\alpha_r = \frac{\nu - br}{s}.$$

Because the rolling effect is ignored, there is no difference between the load on the right and left wheels. Let N_f and N_r are the load on the front and rear wheels. The forces along the y of LC on front and rear wheels are:

$$F_{yf} = -C_{\alpha f}\alpha_f, \;\; \textit{if } N_f\mu/2 > C_{\alpha f}\tan(|\alpha_f|$$

and

$$F_{yr} = -C_{\alpha r}\alpha_r, \;\; \textit{if } N_r\mu/2 > C_{\alpha r}\tan(|\alpha_r|) \; .$$

Otherwise,

$$F_{yf} = \mu N_f \mathrm{Sgn}(\alpha_f)(1 - x_f/2), x_f = N_f\mu/2C_{\alpha f}\tan(|\alpha_f|)$$

and

$$F_{yr} = \mu N_r \mathrm{Sgn}(\alpha_r)(1 - x_r/2), x_r = N_f\mu/2C_{\alpha r}\tan(|\alpha_r|).$$

Let $h = (-(K - MgH_2)\phi - cq - (F_{yf} + F_{yr})H_2)/I_r$. The following represent the five equations of motion:

$$\dot{x} = s\cos\psi - \nu\sin\psi,$$

$$\dot{y} = s\sin\psi + \nu\cos\psi,$$

$$\dot{r} = (F_{yf}a - F_{yr}b)/I_y,$$

$$\dot{\psi} = r,$$

$$\dot{\nu} = (F_{yf} + F_{yr})/M - sr,$$

3.2.2 Trajectory design for a floating spacecraft

Challenging trajectory design problems come from the aerospace domain. For example, computing a re-entry trajectory for the NASA X-33 reusable spacecraft is a formidable task.

In this example, a spacecraft floating in the space can translate and rotate freely. Three thrusters are equipped to the spacecraft. To provide both force and torque, thrusters are designed to make the direction of the force provided by thrusters not pass through the mass center of the spacecraft. The planner needs to design a trajectory to guide the spacecraft from the initial state to a goal state without hitting the obstacles.

Similar to the above problem, two frames, a global frame and a local frame fixed on the spacecraft are chosen. Let M is the mass of the spacecraft. With respect to local frame fixed on the spacecraft, I is its inertial matrix, the forces from the thrusters are F_x, F_y, F_z which are along the direction of x, y, z axises, the vertical distance from the mass center to the force is L_x, L_y, L_z. Fig. 3.3 shows the force position and direction, in which the spacecraft is represented by a cylinder. With respect to the global frame, x, y, z are the position of the spacecraft, ψ, ϕ, β are its Euler orientation angle, s_x, s_y, s_z are the speeds of translation in x, y, z axis directions and s_ψ, s_ϕ, s_β are the speeds of the Euler angles, the orientation transformation matrix is R, which is the function of ψ, ϕ, β. The problem can be formulated as follows:

1. **12-Dimensional State Space:** A state is of form, $(x, y, z, \psi, \phi, \beta, s_x, s_y, s_z, s_\psi, s_\phi, s_\beta)$.

2. **3-Dimensional Input Set:** A input, $u = (F_x, F_y, F_z)$, consists of a discretized input from each thruster. The input set is the combination of all possible inputs.

3. **Equation of Motion:** The twelve motion equations are:

$$\dot{x} = s_x,\ \dot{y} = s_y,$$

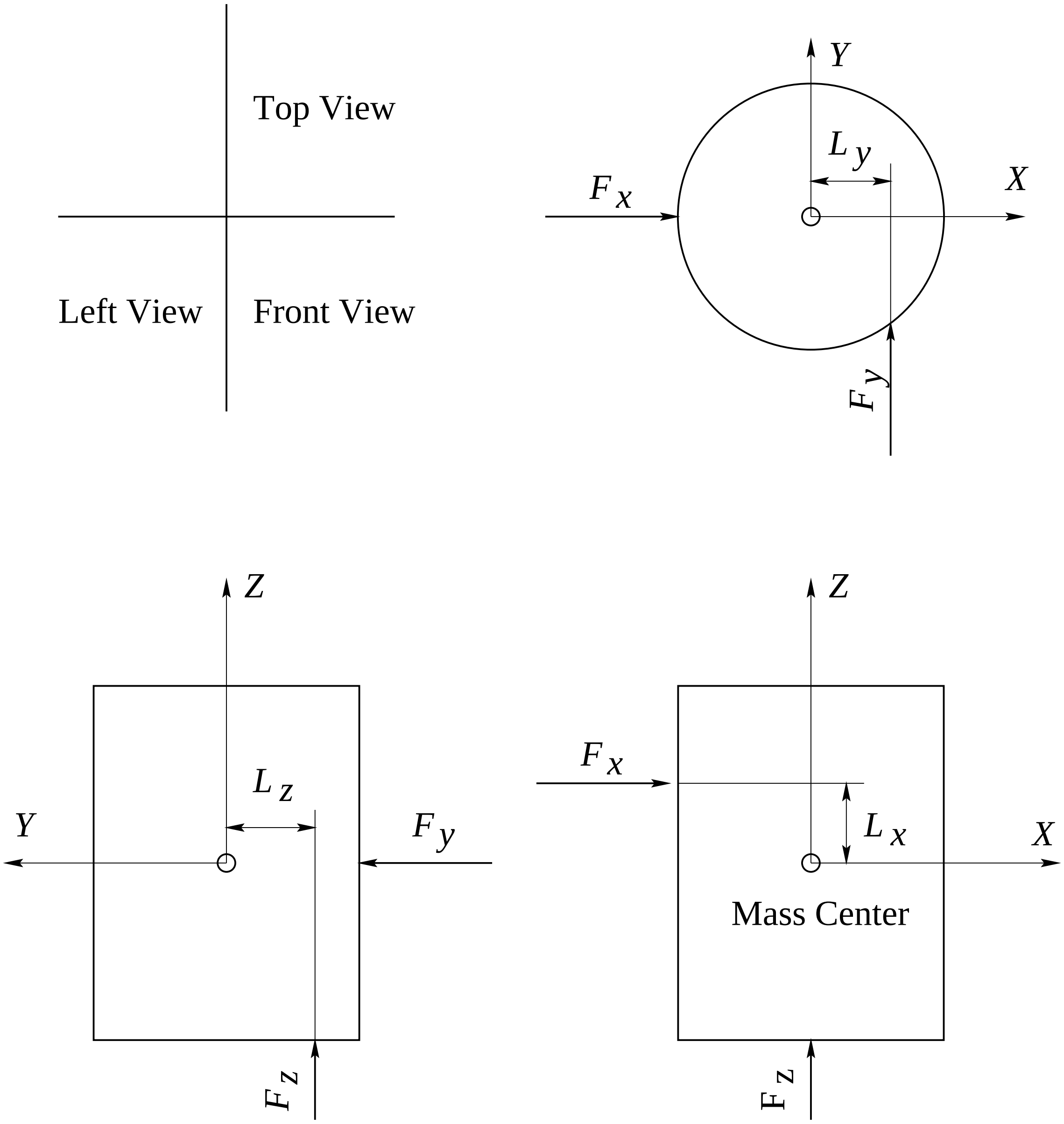


Figure 3.3 Position and direction of the forces applied on the spacecraft.

$$\dot{z} = s_z,\ \dot{\psi} = s_\psi,$$

$$\dot{\phi} = s_\phi,\ \dot{\beta} = s_\beta,$$

$$\dot{s}_x = [1,\ 0,\ 0]\ A_{xyz},$$

$$\dot{s}_y = [0,\ 1,\ 0]\ A_{xyz},$$

$$\dot{s}_z = [0,\ 0,\ 1]\ A_{xyz},$$

$$\dot{s}_\psi = [1,\ 0,\ 0]\ A_{\psi\phi\beta},$$

$$\dot{s}_\phi = [0,\ 1,\ 0]\ A_{\psi\phi\beta},$$

$$\dot{s}_\beta = [0,\ 0,\ 1]\ A_{\psi\phi\beta},$$

in which

$$A_{xyz} = R\ [F_x,\ F_y,\ F_z]^T/M$$

and

$$A_{\psi\phi\beta} = R\ I^{-1}\ [F_z L_z,\ F_x L_x,\ F_y L_y]^T.$$

CHAPTER 4 DESCRIPTION OF THE IMPROVED RRT

Our methods are inspired from the RRT-based planner. In Sections 4.1 and 4.2, we present the RRT and its metric issues before the description of the new methods. In Section 4.3, techniques used to improve the RRT are introduced and Section 4.4 presents the improved RRTs. The framework for planners based on RRTs is presented in Section 4.5.

4.1 Rapidly-Exploring Random Trees

To understand the metric sensitivity problem of the RRT-based planner, it is necessary to first describe the RRT, which is employed by the planner to explore the state space. The RRT is a search graph, $G(N_G, E_G)$, which can explore a high-dimensional state space quickly by using randomized heuristics to guide the search. An RRT is constructed using the algorithm described in Fig. 4.1 and Fig. 4.2, in which NEAREST_NEIGHBOR chooses the nearest state to the random state and CONTROL selects the best input to extend the nearest state. Their algorithms are presented in Fig. 4.3. Initially, x_{init} forms the root of the tree. For each iteration, the algorithm chooses $x_{random} \in X$, and selects $x_{near} \in N_G$ based on a given metric function, ρ. From x_{near}, an input is chosen and applied to generate $x_{new} \in X_{free}$. If the path from x_{near} to x_{new} satisfies the global constraints, and the distance between x_{new} and x_{random} is the smallest in all collision-free states generated by applying every input in the input set to the selected state, this new state will be added to the search tree.

BUILD_RRT(x_{init}, K)
G.init(x_{init});
for $k = 1$ **to** K **do**
    $x_{random} \leftarrow$ RANDOM_STATE();
    $x_{near} \leftarrow$ NEAREST_NEIGHBOR(x_{random}, G);
    $u_{best}, success, x_{new} \leftarrow$ CONTROL(x_{near}, x_{random}, G);
    if success
        G.add_vertex(x_{new});
        G.add_edge($x_{near}, x_{new}, u_{best}$);
Return G, x_{new}

Figure 4.1 The basic RRT construction algorithm.

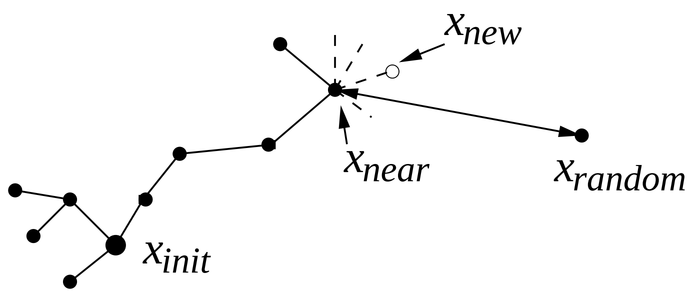


Figure 4.2 RRT extension in one iteration.

An RRT is a combination of the randomization and a Voronoi region bias. A node is extended only when the random states lie in its Voronoi region. When a search tree is used to explore the state space, the frontier nodes, which can explore the space most efficiently and quickly, probably have the biggest Voronoi region. When combined with the random heuristics, the random states will make the frontier nodes have more probability to be extended automatically. Figure 4.4 illustrates how the RRT tends to explore towards bigger Voronoi regions.

4.2 Dependency of RRTs on the Metric

As mentioned above, part of the success of the RRT is based on Voronoi region bias. The metric function is very important to the performance of the RRT because it directly affects the Voronoi regions. The ideal metric would be the optimal cost-to-go, which is

```
NEAREST_NEIGHBOR(x_random, G)
d_min ← ∞
for all x in G
    d ← ρ(x, x_random);
    if d < d_min
        d_min ← d;
        x_best ← x;
return n_best
```

```
CONTROL(x_near, x_random, G)
d_min ← ρ(x_near, x_random);
success ← false;
for all u in U
    x′ ← Integrate(x_near, u);
    if D(x′)
        d ← ρ(x′, x_random);
        if d < d_min
            d_min ← d;
            success ← true;
            u_best ← u;
return u_best, success, x_new
```

Figure 4.3 An algorithm to choose the best input and its resulting new state.

the cost to move from one state to another state under the optimal trajectory. The cost might be the distance traveled, the energy consumed, or the time elapsed during the execution of a trajectory. Unfortunately, calculating the optimal cost-to-go is at least the same difficulty as the trajectory design problem. Both the differential constraints and the global constraints have to be considered. For holonomic path planning problems a Euclidean metric is usually employed, and the RRT-based planners produce good results; however, for problems involving differential constraints, Euclidean metrics provide bad information, which degrades RRT performance. The effect of differential constraints is illustrated in Fig. 4.5. Suppose a forward-only car-like robot is driving forward with high speed, such as 96 kph. The radius of the smallest circle in which it can turn is 100 meters. The robot is driving past the origin of the x axis to the positive direction of the x axis. One state is at 150 meters, and another is at -100 meters. The Euclidean metric

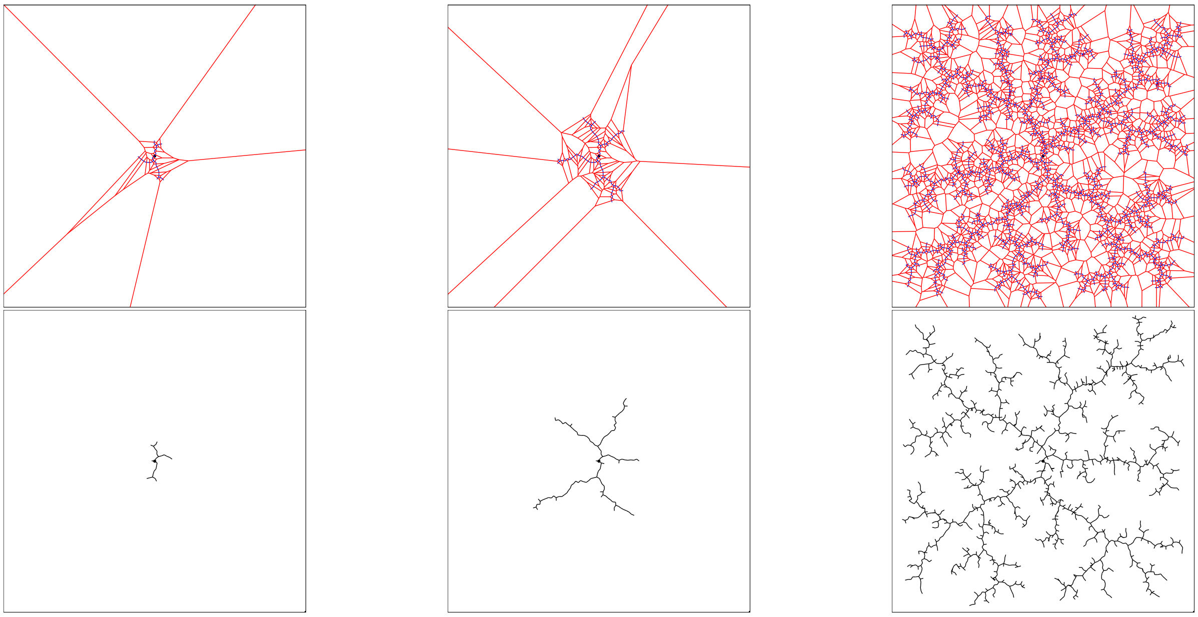


Figure 4.4 The RRT is a Voronoi biased exploration.

might cause the algorithm to prefer the -100 state; however, it is worse because the car cannot drive backwards, and would have to turn around to go to -100 state; the state at 150 is closer in terms of the true cost. To understand the effect of global constraints on the metric, imagine that a robot is in an environment with non-convex obstacles (see Fig. 4.6), in which ROBOT2 is closer to ROBOT3 than ROBOT1. Two states might be close in terms of a Euclidean metric, but the correct metric should use the shortest path within the labyrinth. This is a well-known problem, which also occurs in potential field methods [3, 44].

The performance degradation occurs for the following reasons:

1. The RRT chooses the nearest state using only the metric function. If the metric function provides bad information, the tree might grow in the wrong direction, which leads to inefficient exploration. Another problem is that many states might be chosen numerous times for expansion, even though they are either destined to a result in a collision or all of their successors have been already generated. The combination of these two problems makes the probability of finding the solution quickly decrease greatly.

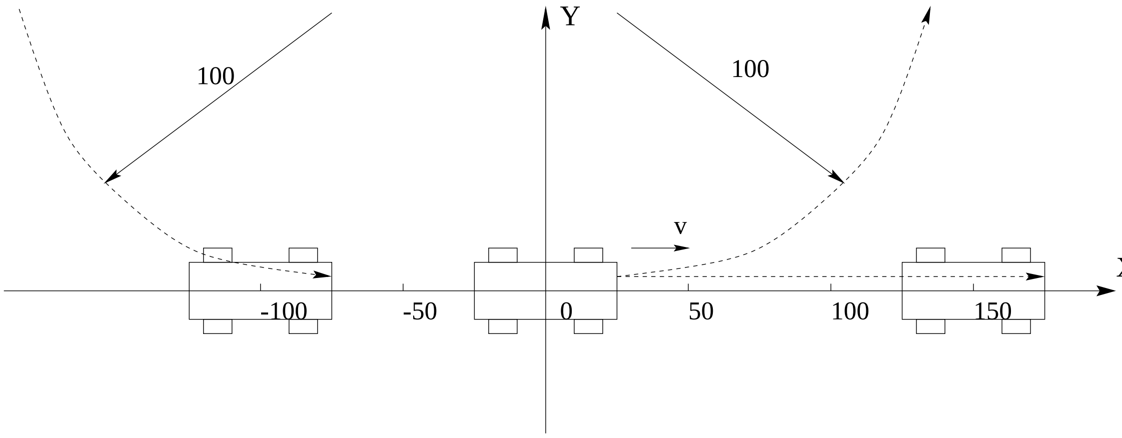


Figure 4.5 The metric problem with differential constraints.

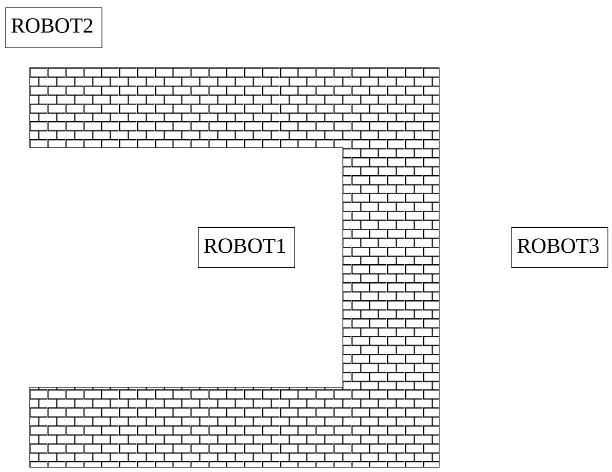


Figure 4.6 The metric problem with global constraints.

Figure 4.7 shows a snapshot of an RRT and corresponding Voronoi regions for the forward-only car-like robot, which is the same robot as that in Fig. 4.5. It is clear that the Euclidean metric provides poor information because the region just behind the current point is considered the nearest region. Suppose state x_1 cannot be extended to the goal state because the path extended from it will inevitably run into the collision region. The solution have to be extended from state x_2. Assume $V(x_2)$ is the Voronoi region corresponding to x_2, if we only choose x_{near} according to the metric function, x_2 can only be chosen when the random state is in $V(x_2)$. Even choosing $x_{random} = x_{goal}$ cannot increase the probability to extend x_2.

2. The input, u_{best}, is selected by relying only on the metric. The original CONTROL algorithm (Fig. 4.3) uses Voronoi regions of the successors of x_{near} to choose u_{best}. The successor, whose Voronoi region includes x_{random}, will be appended to the search tree. Similar to the above situation, this operation might drive the robot along a poor path. The input that yields good exploration might be discarded because the new state derived from this input has a 'larger' distance to the random state than that of the other states derived from the other inputs. Another problem is that the same input can be chosen several times even though it has already been applied and generated a new state. In Fig. 4.8, x_{21}, x_{22}, and x_1 are successors of x_2, and $V(x_{21}), V(x_{22})$, and $V(x_1)$ are their corresponding Voronoi regions. Suppose only x_{21} can lead to the goal. The random state must lie in $V_{x_2} \cap V_{x_{21}}$, which makes the probability of finding the solution smaller.

4.3 Adaptive Reduction of Metric Sensitivity

Designing a good metric is a difficult problem and requires much knowledge in the problem domain. In fields with throughout research, a good metric can be designed and leads to good performance in an RRT-based planner [29]; however, in many active research fields, a good metric is hard to design such that RRT-based planners will show

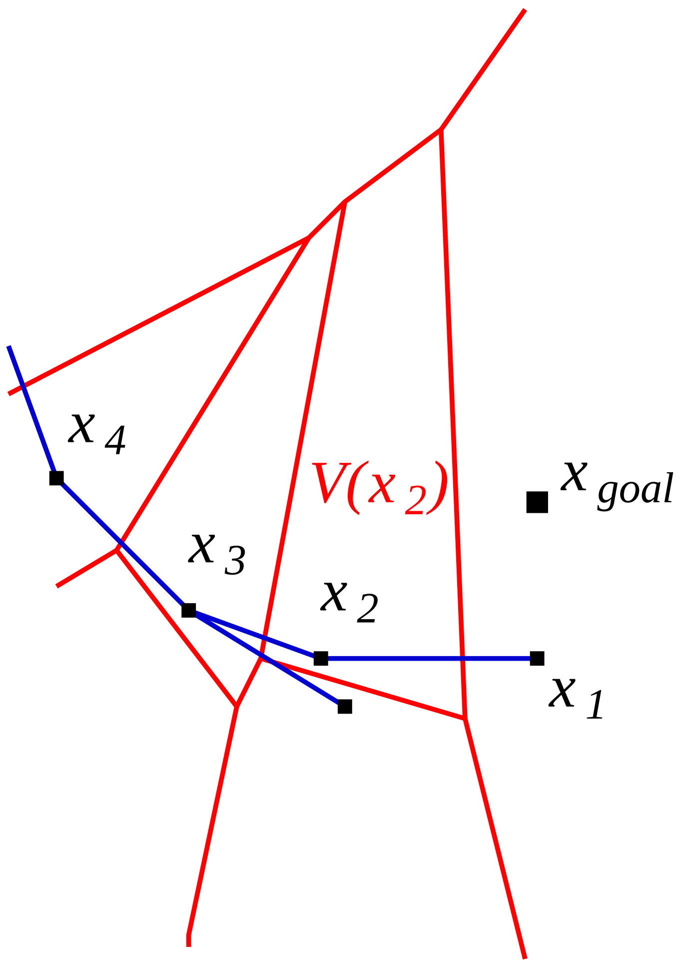


Figure 4.7 Metric sensitivity problem of the RRT in choosing the nearest neighbor.

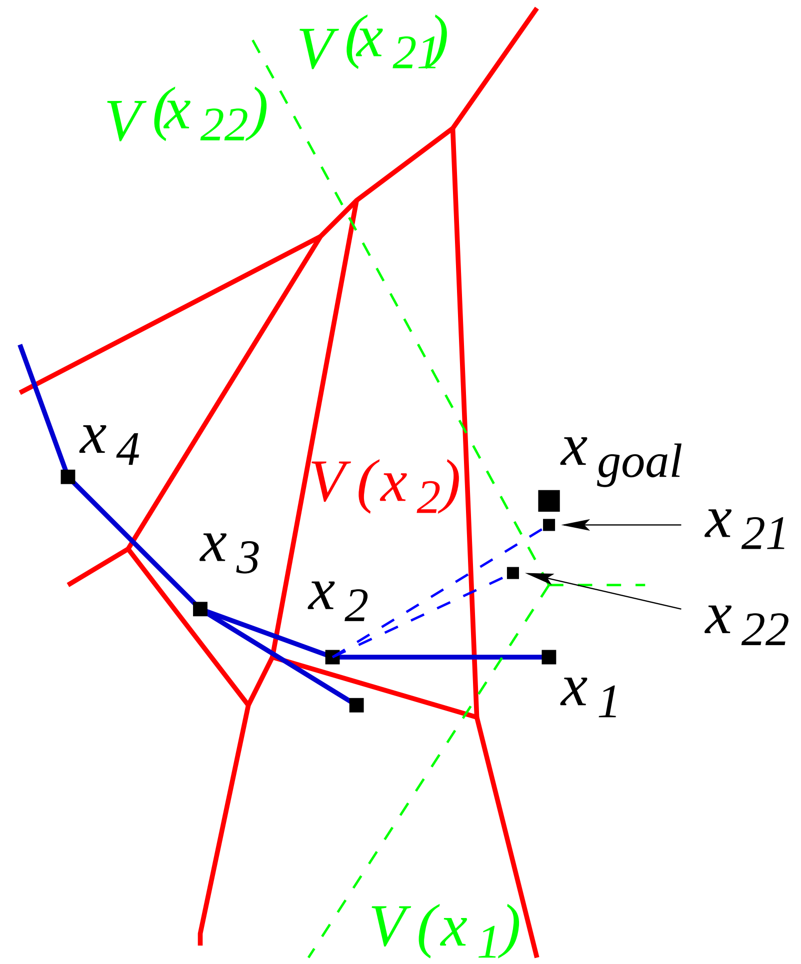


Figure 4.8 Metric sensitivity problem of the RRT in choosing the best input.

poor performance in these fields. In this thesis, a method alleviates the dependency on the metric by applying information collected during the exploration, which makes the RRT less metric-sensitive and more robust.

According to the metric problems mentioned above, the following information is collected during the search:

1. **Exploration information:** For each RRT node, we record whether an input has already been applied and evaluated. If an input has been applied for a state, it will not be considered for the state again. If the inputs for a state are EXPANDED, this state will be excluded from the search space. In this way, the RRT will be more likely to explore the unexplored state space. Moreover, the planner avoids doing collision checking for the same input and state repeatedly.

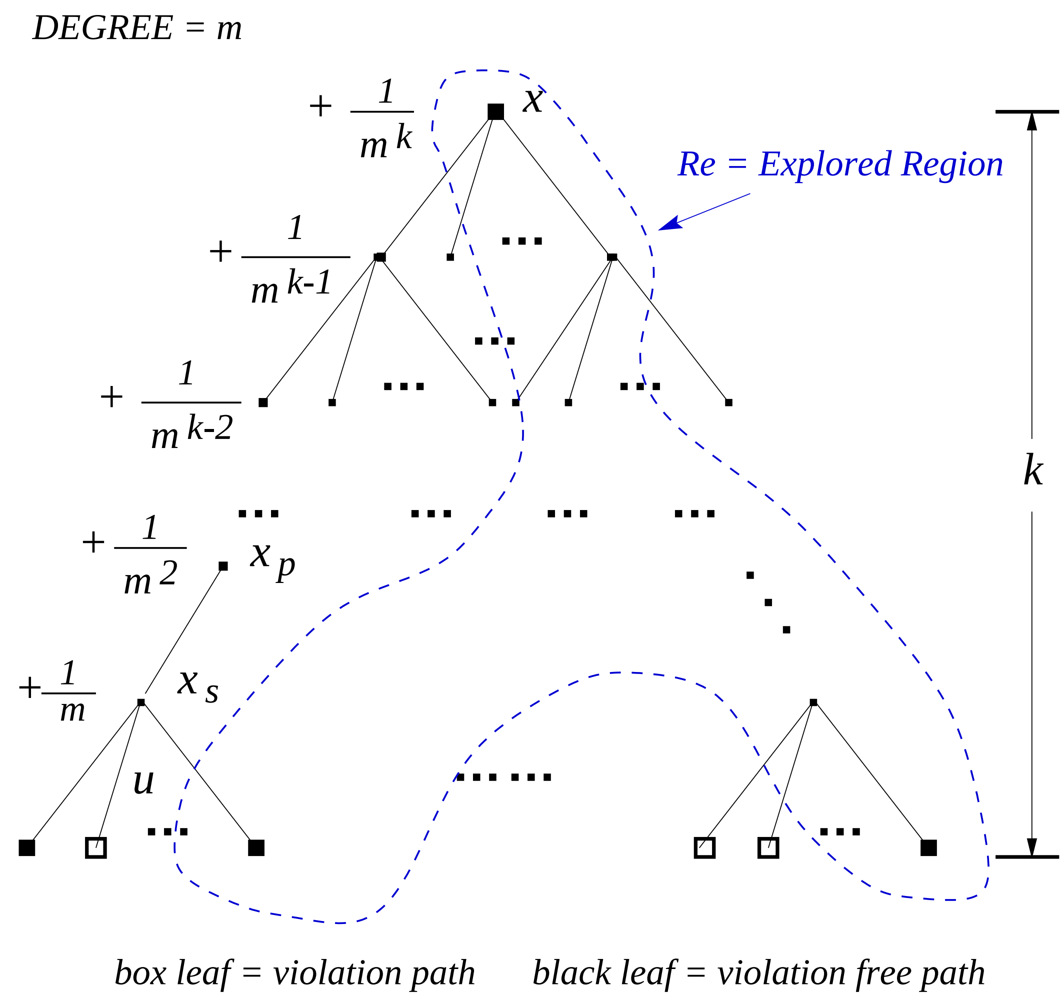
2. **Constraint violation frequency:** Refer to Fig.4.9. Given a set, S, of all input sequences of length k time steps, the *constraint violation probability* (CVP) of the state, x, is the probability of the path which is generated by applying a random input sequence in S to x and violates the constraints. It can be calculated by applying all input sequences in S to x and dividing the number of collision paths by the number of all input sequences. CVP provides the distribution of global state constraints in the state space. States with CVP $= 1$ will be prevented from expanding because all paths via them will collide. States with less CVP will be given more priority to expand because they are more likely to evade the constraints. However, calculating the actual CVP is impractical for large input sets and long input sequences (one might as well use dynamic programming to solve the problem in this case). The constraint violation frequency, which is always smaller than the actual probability (explained in the following part), is used in this thesis. With more and more exploration, the constraint violation frequency will approach but never exceed the actual probability. Any states along a solution path have a

strictly positive probability to be chosen for extension. The proof is in Lemma 1 in Chapter 5.

To collect the exploration information, a vector, ω, is kept in each node in the search graph. Each element of ω corresponds to an input in U. Initially, each element of ω is set to be UNEXPANDED. If one input leads to a collision or it successfully expands a state and generates a new node in the tree, its corresponding element in the vector is set to be EXPANDED.

To calculate the constraint violation frequency, the following method is adopted. Initially, for each new state appended to the search tree, its violation frequency is 0. In Fig. 4.9, suppose there are m inputs in U, when the new state, x_s, is selected for extension, instead of trying all of the input sequences of length k to calculate the real violation probability, we apply these m inputs as the first step of all input sequences. If input $u \in U$ leads x_s to collision directly, the constraint violation frequency of x_s will be increased by $\frac{1}{m}$. Furthermore, the constraint violation frequency of its parent state, x_p, is increased by $1/m^2$ because we are sure that at least one input of one of its child states leads to eventual collision. Similarly, for the k^{th} parent state of x_s, one collision input of x_s will increase its path violation frequency by $1/m^{(k+1)}$. Because the constraint violation frequency accumulates only when collisions happen and the RRT does not perform an exhaustive search, only part of all possible collision paths are considered; the constraint violation frequency will always be no larger than the actual probability. Figure 4.10 shows the value of the constraint violation frequency collected in the exploration and provides the global constraints information in the searched state space, in which the points represent the states and the shade of a point represents the value of the constraint violation frequency (darker indicates higher violation probability).

The exploration information and constraint violation frequency can help the RRT to choose better states for expansion. The modified NEAREST_NEIGHBOR function



$$\textit{Constraint Violation Probability of } x = \frac{\textit{Number of all box leaves}}{\textit{Number of all the leaves}}$$

$$\textit{Constraint Violation Tendency of } x = \frac{\textit{Number of box leaves in Re}}{\textit{Number of all the leaves}}$$

Figure 4.9 Constraint violation probability and constraint violation frequency.

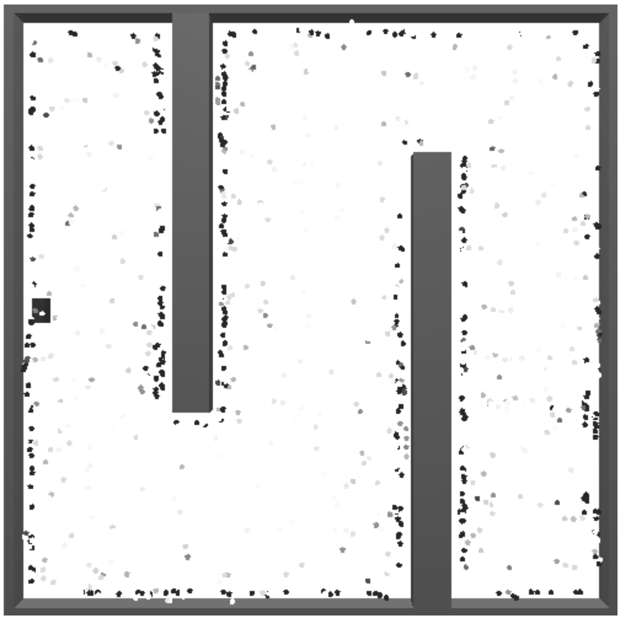


Figure 4.10 Constraint violation frequency collected in the exploration.

is given in Fig. 4.11. When selecting the nearest neighbor state for a random state, the planner will first check if all the inputs of a state, x, are EXPANDED. If they are EXPANDED, x is just ignored; otherwise, the probability of not choosing x equals constraint violation frequency of x. Even if there are some states with UNEXPANDED inputs, the constraint violation frequency might be low enough to prevent them from being chosen. It will waste one iteration to explore the state space. Under this condition, the state with the smallest distance to the random state will be selected. In Fig. 4.7, in which $\sigma(x)$ represents the constraint violation frequency of a state x, if all successors of x_1 are EXPANDED, x_1 will not be considered and $\mu(V(x_2))$ will increase and be more likely to be chosen for extension. When x_2 is also impossible on the solution path, after several failed attempts its constraint violation frequency will increase such that x_3 and x_4 might extend earlier.

The exploration information is also helpful for selecting the best input. The modified CONTROL algorithm is shown in Fig. 4.12, in which the UPDATE_TREEINFO (in Fig. 4.13) function updates the information during the exploration. When a state is selected,

NEAREST_NEIGHBOR(x_{random}, G)
$d_{min} \leftarrow \infty$;
$d_{min'} \leftarrow \infty$;
for all x in N_G
    if $\exists u$ of x are not EXPANDED
        $d \leftarrow \rho(x, x_{random})$;
        if $d < d_{min'}$
            $d_{min'} \leftarrow d$;
            $x_{best'} \leftarrow x$;
        $r \leftarrow$ random number in [0,1];
        if $r > \sigma(x)$
            if $d < d_{min}$
                $d_{min} \leftarrow d$;
                $x_{best} \leftarrow x$;
if $d_{min} \neq \infty$
    return x_{best};
else
    return $x_{best'}$;

Figure 4.11 The modified NEAREST_NEIGHBOR algorithm.

whether an input is EXPANDED will be checked first. If it is EXPANDED, it will not be considered; otherwise, it will be extended to determine whether the new state is collision free and is closer to x_{random}. If this new state is collision free and closer to random state, this input will be considered as a better input. If the new state is in the collision region, the exploration information and constraint violation frequency information will be updated. Refer to Fig. 4.8. When x_2 is chosen as the nearest state, x_{22} can only be considered once. After it is generated, x_{21} will be more likely to be extended.

Using the original BUILD_RRT algorithm, the exploration information only partially alleviates the problem of repeated states in the exploration because of repeated states from cyclic paths. For example, for a forward-only car-like robot with constant speed, the robot may traverse a circle to come back to its original state. This repeated state cannot be avoided by using the exploration information. It leads a cycle that which prevents the

```
CONTROL(x_near, x_random, G)
  d_min ← ∞;
  success ← false;
  for all u in U
      if u has not been EXPANDED
          x' ← Integrate(x_near, u);
      if D(x')
          d ← ρ(x', x_random);
          if d < d_min
              d_min ← d;
              success ← true;
              u_best ← u;
      else
          mark u as EXPANDED
          UPDATE_TREEINFO(x_near, G)
  mark u_best as EXPANDED
  return u_best, success, x_new
```

Figure 4.12 Modified CONTROL algorithm.

```
UPDATE_TREEINFO(x_near, G)
  p ← 1/m;
  σ(x_near) ← σ(x_near) + p;
  p ← 1/m^2;
  x_1 ← x_near;
  while x_1 ≠ x_init
      x_2 ← parent(x_1);
      σ(x_2) ← σ(x_2) + p;
      p ← p/m;
      x_1 ← x_2;
```

Figure 4.13 An algorithm to update the constraint violation frequency information.

algorithm from terminating even when it has explored all of the state space. Thus if it is impossible to guarantee there are no cyclic paths, the original BUILD_RRT needs the maximum depth to guarantee its completeness. The maximum depth can be determined in some problems which have restrictions on the maximum time, cost, or energy.

4.4 Extensions of Improved RRTs

For the problems with cyclic paths and no restrictions on the maximum cost of the path, we modify the BUILD_RRT algorithm by applying a tighter procedure to exclude repeated states in Fig. 4.14, in which besides the exploration information, a neighborhood is used to exclude the repeated states. Besides using the exploration information, a small neighborhood is assigned to each node in the search graph. To add a new node, we need to check whether the new node is in neighborhoods of existing nodes. If it is in some existing neighborhood, it will be discarded because its region has already been explored; otherwise, it will be added to the RRT . In this way, repeated states are excluded.

4.4.1 Using neighborhoods to exclude repeated states

The neighborhood can be in many different forms, which can lead to the same complete conditions (Corollary 1 in Chapter 5). In our implementation, a ball region, $B(x, r_b) = \{x' \mid \rho(x, x') < r_b, x' \in X\}$, is used to represent the neighborhood of $x \in X$. If the system transition equation satisfies the Lipschitz condition, and if the Lipschitz constant and maximum search depth can be determined, r_b can be determined according to Theory 1 in Chapter 5; otherwise, to maintain the expansion of the search graph, r_b has to be small enough such that none of the successors of any state lie in the same ball region. Because each input will be applied for a fixed time length, δt, the minimum

BUILD_RRT_VARIANT1(x_{init}, r_b, K)

G.init(x_{init});
for $k = 1$ **to** K **do**
  $x_{random} \leftarrow$ RANDOM_STATE();
  $x_{near} \leftarrow$ NEAREST_NEIGHBOR(x_{random}, G);
  $u_{best}, success, x_{new} \leftarrow$ CONTROL(x_{near}, x_{random}, G);
  if success and $\forall x \in N_G, x_{new} \notin B(x, r_b)$
    G.add_vertex(x_{new});
    G.add_edge($x_{near}, x_{new}, u_{best}$);
Return G, x_{new}

Figure 4.14 The first variant of the BUILD_RRT algorithm.

length over all possible expansions can be calculated as follows:

$$L_{\delta t} = \min_{u \in U, x \in X} \rho(x_{u,\delta t}, x),$$

in which $x_{u,\delta t}$ is the state generated by applying input u to state x with fixed time, δt. The r_b can be chosen as any value between 0 and $L_{\delta t}$. As r_b decreases, the number of the nodes in the search tree and the time to exclude the repeated states will increase. The calculation of $L_{\delta t}$ is a potentially difficult optimization problem. In many cases, some methods can be used to approximate it, such as calculating $\rho(x_{u,\delta t}, x)$ for some randomly chosen states and inputs, and choosing the least value as $L_{\delta t}$.

4.4.2 Returning an optimal solution in the explored state space

Classical dynamic programming explores all the state space and returns an optimal solution; however, randomized methods sacrifice the global optimality by only exploring part of the state space. Another variant of BUILD_RRT is given in Fig. 4.16 to return an optimal path in the explored region. Assume the current minimum path cost, $CMC(x)$, is stored at each state, $x \in G$. In Fig. 4.15, if x_{new} is successfully generated from its parent state, x_p, the following procedure will be taken: If $\forall x \in N_G, x_{new} \notin B(x, r_b)$, it will be added to the graph, and its path cost $CMC(x_{new}) = CMC(x_p) + C(x_p, u, \delta t)$, in which $u \in U$ is the input leading x_p to x_{new} and $C(x_p, u, \delta t)$ is a cost function; otherwise, if $x_{new} \in B(x_1, r_b)$ and $CMC(x_{new}) \geq CMC(x')$, x_{new} will be discarded and the graph is unchanged. If $CMC(x_{new}) < CMC(x_1)$, the x_{new} replaces node x_1 in G, the link between x_p and x_{new} is added, and the link between x_2 and x_1 is removed. Under these procedures, if a solution is found, it will be the minimum cost path in the explored state space.

4.5 Design of Improved Planners

The basic RRT attempts to rapidly explore the state space. To solve a trajectory design problem, RRTs have been adapted and incorporated into a planning algorithm in [61]. The improved RRT indicated in this thesis can be incorporated into any of the RRT-based planners by modifying the following termination conditions and enhancement techniques.

4.5.1 Deterministic termination condition

To stop searching after all of the space is explored, the termination condition should be determined. For the original RRT, infinite number of iterations is required to reach a probabilistic completeness [61]. The improved RRTs combine the random techniques and classical dynamic programming such that deterministic termination conditions can be used for improved RRT-based planners.

The improved RRT that uses exploration information and constraint violation frequency allows repeated states resulting from cyclic paths; hence it is only suitable for problems which have a finite search graph and no cyclic paths, or problems with maximum search depth, k_{max}. For a given δt and τ, if all the states, either before depth k_{max} or in X, accessible from x_{init} are explored and no solution is found, we say there is no

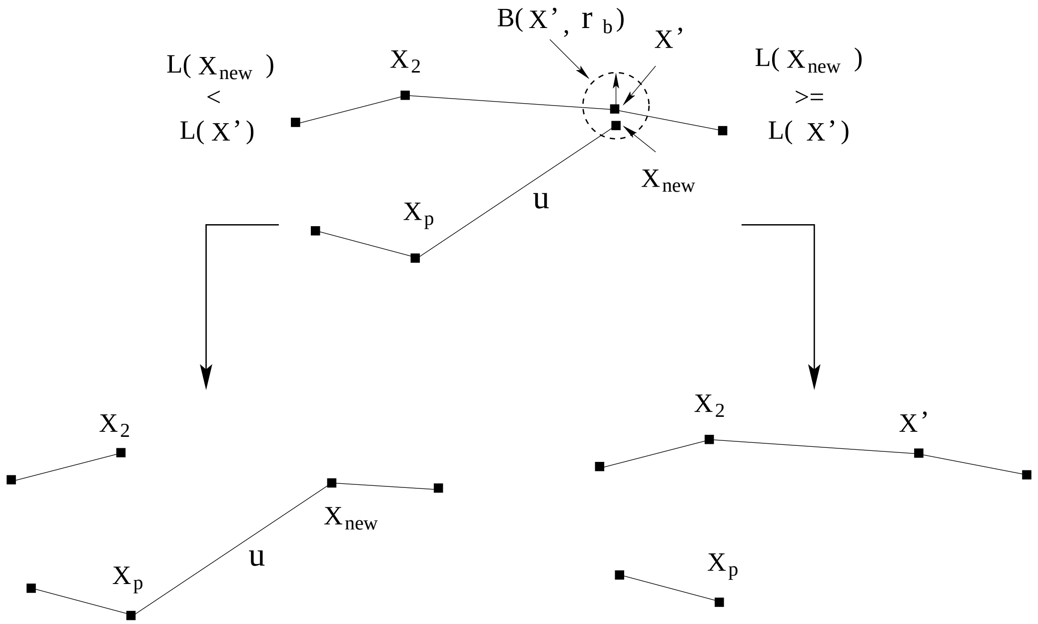


Figure 4.15 The method to keep the minimum cost path.

```
BUILD_RRT_VARIANT2(x_init, r_b, K)
G.init(x_init);
for k = 1 to K do
    x_random ← RANDOM_STATE();
    x_near ← NEAREST_NEIGHBOR(x_random, G);
    u_best, success, x_new ← CONTROL(x_near, x_random, G);
    if success
        if ∀x_1 ∈ N_G, x_new ∉ B(x_1, r_b)
            G.add_vertex(x_new);
            CMC(x_new) = CMC(x_near) + c(x_near, u_best, δt)
            G.add_edge(x_near, x_new, u_best);
        else
            CMC(x_new) = CMC(x_near) + c(x_near, u_best, δt)
            if CMC(x_new) < CMC(x_1)
                x_1 = x_new
                G.add_edge(x_near, x_new, u_best);
                G.remove_edge(x_2, x_1), in which x_2 is parent node of x_1;
Return G, x_new
```

Figure 4.16 The second variant of the BUILD_RRT algorithm.

solution at a resolution given by δt and τ.

Because extensions of the improved RRT use neighborhoods to discretize X, it can be applied to any trajectory problems. Exploring all the accessible nodes from x_{init} achieves a *resolution completeness* (refer to Theorem 1 in Chapter 5).

4.5.2 Enhancement techniques

Goal-biased and bi-directional search have been employed to enhance RRT-based planners in [61]. The first method will not be affected by the metric problem, while the second technique needs modification because of the metric problem.

The first method is to choose the goal state as the random state with probability, β_{goal}. Refer to 4.7, if $\beta_{goal} = 0$, then the RRT will uniformally extend to all of the directions such that it is easy to avoid local minimum but has no bias to extend towards the goal state. If $\beta_{goal} = 1$, the improved RRT turns out to be a greedy search considering

```
RANDOM_STATE(β_goal )
γ ← Random(0, 1);
if γ ≥ β_goal
    return RANDOM_STATE();
else
    return x_goal;
```

Figure 4.17 The modified RANDOM_STATE algorithm.

repeated states. Even though it always extends toward the goal state, it is easy to get trapped in the local minimum. According to our experiences, $\beta_{goal} \in [0.05, 0.2]$ yields good performance for most problems. The RANDOM_STATE function with goal-bias β_{goal} is shown in Fig. 4.17, in which $Random(0, 1)$ is a function which returns a real number in $[0, 1]$ and with probability β_{goal}, x_{goal} will be chosen as the random state.

The second technique borrows the idea from the AI bi-directional search [40]. Two search graphs, G_1 and G_2, are extended from x_{init} and x_{goal}, respectively. If $\exists x_1 \in G_1, x_2 \in G_2, \rho(x_1, x_2) < \tau$, a solution is declared. Because it will be more likely to connect two search trees to find a solution, its performance is generally better in comparison to a single-tree planner; however, detecting a solution becomes a challenging problem. In each iteration, the original bi-directional RRT-based planners only check if the two new states generated from both search structures is close enough for a solution. It is possible that during the exploration of the dual RRTs, the original planner might continue to explore the state space even though a solution already exists. This problem is caused by the frontier of two search structures passing through each other, which is mentioned in [40]. To overcome the above problem, we test whether the new node in one tree is in the tolerance region of any node in the other tree. Although costly, it generally leads to reliable performance because all alternatives are considered. The solution checking procedure is in Fig. 4.18.

Combining different determination conditions and acceleration techniques, different

DUAL_TREE_SOLUTION_CHECK(x_1, x_2, G_1, G_2)
forall $x \in G_1$
  if $\rho(x, x_2) < \tau$;
    return true;
forall $x \in G_2$
  if $\rho(x, x_1) < \tau$;
    return true;
return false;

Figure 4.18 Solution check algorithm for the bi-directional RRT-based planner.

SINGLE_RRT_PLANNER(X, x_{init}, x_{goal}, δt, ρ, r_b, τ)
G.init(x_{init});
while *termination condition is not satisfied*
  $G, x_{new} \leftarrow$ BUILD_RRT(1);
  if $x_{goal} \in B(x_{new}, r_b)$;
    return solution path;

Figure 4.19 The framework for the single directional RRT-based planner.

planners can be derived. The framework of the single and dual directional planners are given in Fig. 4.19 , in which BUILD_RRT(1) represents extending an RRT for one iteration and returning the new state, and Fig. 4.20, in which BUILD_RRT(G_1, G_2, 1) represents extending both RRTs for one iteration using an RRT variant and x_{new1}, x_{new2} represent two new nodes in both RRTs.

DUAL_RRT_PLANNER(X, x_{init}, x_{goal}, δt, ρ, r_b, τ)
G_1.init(x_{init});
G_2.init(x_{goal});
while *termination condition is not satisfied*
  $G_1, G_2, x_{new1}, x_{new2} \leftarrow$ BUILD_RRT(G_1, G_2, 1);
  if DUAL_TREE_SOLUTION_CHECK($x_{new1}, x_{new2}, G_1, G_2$)
    return solution path;

Figure 4.20 The framework for the bi-directional RRT-based planner.

CHAPTER 5 ALGORITHM ANALYSIS

In Chapter 4, improved RRTs and planners have been introduced. Some analysis will be done in this chapter. In Section 5.1, an accessibility graph is defined, from which a resolution completeness theory is derived in Section 5.2. The comparison between the new methods and original RRTs is described in Section 5.3. Section 5.4 presents the worst-case analysis of new RRTs.

5.1 The Accessibility Graph

The accessibility graph describes the connectivity among all of the accessible states from x_{init}. It is the basis of the analysis in the chapter. Before introducing the accessibility graph, the following definitions are given:

Definition 1 *Trajectory design problem, Δ:*

It is composed of $(X, D, x_{init}, X_{goal}, U, \delta t, \rho, \tau, f)$, which are defined in Chapter 3.

Definition 2 *Solution path, π:*

It is a function $\pi : [t_0, t_f] \to X_{free}$ (defined in Chapter 3). Let $u(t)$ be the solution control function (defined in Chapter 3); we also represent π by either an input sequence $\pi_u = \{u_1, u_2, \cdots, u_K\}$ in which $u_i = u(t_0 + (i-1)\delta t)$ or a state sequence $\pi_x = \{\ x_0 = x_{init}, x_1, \cdots, x_K\}$ in which $x_i = \pi(t_0 + i\delta t)$.

Definition 3 *Exact solution oracle, $EO(\Delta, x, x')$:*

$EO(\Delta, x, x')$ indicates whether $\exists \pi$ such that $\pi(t_0) = x$ and $\pi(t_f) = x'$ for a given Δ.

$$EO(\Delta, x, x') = \begin{cases} true, & \text{if } \exists \pi \text{ for } \Delta \text{ such that } \pi(t_0) = x \text{ and } \pi(t_f) = x', \\ false, & \text{otherwise.} \end{cases} \tag{5.1}$$

Definition 4 *Approximate solution oracle, $AO(\Delta, x, x')$:*

$AO(\Delta, x, x')$ indicates whether $\exists \pi$ such that $\pi(t_0) = x$ and $\rho(\pi(t_f), x') < \tau$ for a given Δ, in which τ is the solution tolerance.

$$AO(\Delta, x, x') = \begin{cases} true, & \text{if } \exists \pi \text{ for } \Delta \text{ such that } \pi(t_0) = x \text{ and } \rho(\pi(t_f), x') < \tau, \\ false, & \text{otherwise.} \end{cases} \tag{5.2}$$

Definition 5 *System transition equation, $s(u, x, t, \delta t)$:*

It is a function: $s : U \times X \times [t_0, t_f] \times (0, \infty) \rightarrow X$. The $s(u, x, t, \delta t)$ calculates the new state by applying a constant input $u \in U$ on a state $x(t) \in X$ for a constant time δt. The $s(u, x, t, \delta t) = x + \int_t^{t+\delta t} f(x, u, t)dt$ without considering the constraints.

Definition 6 *Violation-free system transition equation, $\tilde{s}(u, x, t, \delta t)$:*

It is a function: $\tilde{s} : \tilde{U} \times X_{free} \times [t_0, t_f] \times (0, \infty) \rightarrow X_{free}$, in which $\tilde{U}$ is a function of $x \in X$ and $\forall u \in \tilde{U}$and $x \in X_{free}, s(u, x, t, \delta t) = x + \int_t^{t+\delta t} f(x, u, t)dt$ and $\forall t' \in (t, t + \delta t], x + \int_t^{t+t'} f(x, u, t)dt \in X_{free}\}$. It is a system transition equation considering the global constraints. The $\tilde{s}(u, x, t, \delta t)$ actually is same as $s(u, x, t, \delta t)$ except there are the constraints on u and x and its result value.

Definition 7 *Accessibility graph, G_∞:*

For a given Δ, G_∞ is a graph $G_\infty(N_\infty, E_\infty)$, in which $N_\infty = \{x \mid x \in X, EO(\Delta, x_{init}, x) = true\}$ and $E_\infty = \{e(x, x') \mid x, x' \in N_\infty and \; \exists u \in U such \; that \; x' = \tilde{s}(u, x, t, \delta t)\}$.

Definition 8 *Graph search planners, Γ:*

The class of all planners for a given Δ, in which a graph $G_{sub}(N_{sub}, E_{sub}) \subseteq G_\infty$ will be generated.

Definition 9 *Resolution complete RRT, RC-RRT:*

RC-RRT is a Γ planner and has three variants. Let RC-RRT$_1$ (Fig. 4.1) denote the improved RRT using exploration information and constraint violation frequency, RC-RRT$_2$ (Fig. 4.14) denote the method using $B(x, r)$ in addition to the above information to discretize X, and let RC-RRT$_3$(Fig. 4.16) be the method returning the minimum cost path in the explored state space. All of RC-RRTs use the modified NEAREST_NEIGHBOR(Fig. 4.11) and the modified CONTROL (Fig. 4.12).

The above definition of the accessibility graph, G_∞, was defined through an oracle, which can be applied for any given Δ, but hides the detailed information about G_∞ for a specific problem. To show G_∞ for a specific problem, such as the trajectory design problem considered in the thesis, the construction of G_∞ is presented as follows.

The trajectory design problem, Δ, can be thought of as a discrete-time and discretized input space problem which corresponds to a multi-stage process. Each stage corresponds to a discrete time and has a set of accessible states. To define the multi-stage process precisely, we give the following definition:

Assume the system begins at time t_0. Let stage k correspond to time $t_0 + (k-1)\delta t$, and let N_k be the accessible state set at stage k. We get the following sequence:

Stage 1,

$$N_1 = \{x \mid x = x_{init}\};$$

Stage 2,

$$N_2 = \{x \mid x = \tilde{s}(u, x', t_0, \delta t), u \in \tilde{U}(x'), x' \in N_1\};$$

Stage k,

$$N_k = \{x \mid x = \tilde{s}(u, x', t_0 + (k-1)\delta t, \delta t), u \in \tilde{U}(x'), x' \in N_{k-1}\}.$$

Based on the above sequence, we define $G_\infty(N_\infty, E_\infty)$, in which

$$N_\infty = \bigcup_{k=1}^{\infty} N_k$$

and

$$E_\infty = \{e(x, x') \mid x, x' \in N_\infty, \exists u \in \tilde{U}(x), x' = \tilde{s}(u, x, t, \delta t)\}.$$

If $\exists x_i \in N_i$, and $\exists x_j \in N_j$ such that $i \neq j$ and $x_i = x_j$, then we say G_∞ has *cyclic paths* and the corresponding problem, Δ, is *cyclic*; otherwise, Δ is *acyclic*.

If $\exists m > 1$ such that $N_m - \bigcup_{k=1}^{m-1} N_k = \varnothing$, then N_∞ is finite; otherwise, N_∞ is infinite.

From the above description, it is clear that once Δ is given, G_∞ is fixed. If there exists a solution path from x_{init} to x_{goal} under the control period δt and tolerance τ, it must exist in G_∞. For a given Δ, if no computation error is considered, then no matter which Γ planner is employed to solve the problem, G_{sub} generated by any Γ planner is a subset of G_∞.

Definition 10 *For a given Δ, a state $x \in X_{free}$ is Δ locally controllable if and only if $\exists r > 0$ such that $\forall x', \rho(x, x') < r, EO(\Delta, x, x') = true$.*

To ensure that a Γ planner can terminate in finite time, N_∞ should be finite. If $\exists x \in X_{free}$ such that x is Δ *locally controllable*, then N_∞ will be infinite. When N_∞ is infinite, X has to be discretized such that N_{sub} generated by the planner is finite. This is accomplished in RC-RRT$_2$. Because RC-RRT$_2$ uses neighborhoods to discretize X, it is apparent that the resolution of the discretization will affect the completeness of the algorithm. It needs to be small enough to avoid missing a solution for a given tolerance. If there exists a computation error, the discretization resolution should be big enough to avoid returning a wrong solution. We call a method *resolution complete* if the completeness of the method only holds for some resolution. Resolution completeness of RC-RRTs is discussed in Section 5.2. It can easily be extended to prove the completeness of other Γ planners.

5.2 Resolution Completeness

RC-RRTs use the exploration information and $B(x, r)$ to exclude repeated states, and the constraint violation frequency to make better choices. First, we will prove that the constraint violation frequency will not prevent a solution path from being explored. Then the resolution completeness arguments will be presented.

Lemma 1 *For a motion planning problem, Δ, suppose there exists a path, π, from x_{init} to $x_{goal} \in X_{goal}$, for any state, $x \in \pi_x$, RC-RRT using the constraint violation frequency will choose it with a strictly positive probability in a finite number of iterations.*

Proof: To find the solution $\pi_x = \{x_0 = x_{init}, x_1, \cdots, x_K\}$, all of the nodes in π_x except x_K have to be chosen and extended such that a solution path can be generated. Suppose after l iterations, every $x_i \in \pi_x$ for $0 < i < K$ exists in N_{sub} and $x_i, x_{i+1}, \cdots, x_K$ are still unexplored. Let $p(x_i)$ denote the probability of choosing x_i as the nearest node to extend, $\mu(S)$ be the Lebesque measure defined on X, $V(x_i)$ be the Voronoi region of x_i corresponding the given metric, $\sigma(x_i)$ be the constraint violation frequency, and let m be the number of inputs in U. According to RC-RRT algorithm description (Fig. 4.11), $p(x_i) = \frac{\mu(V(x_i))}{\mu(X)}(1 - \sigma(x_i))$. To show that $p(x_i)$ is a strictly positive number, we first find the lower bound for $\frac{\mu(V(x_i))}{\mu(X)}$.

In Section 5.1, to make the algorithm terminate after a finite number of iterations, N_{sub} is finite such that $\exists \iota > 0$, such that $\forall n \in N_{sub}, \mu(V(n)) > \iota$. Because $\mu(X)$ is also a constant, $\exists \kappa > 0$, such that $\forall n \in N_{sub}, \frac{\mu(V(x_i))}{\mu(X)} > \kappa$.

Secondly, we find a lower bound for $(1 - \sigma(x_i))$. Initially, when x_i is added to G_{sub}, $\sigma(x_i) = 0$. With more iterations, $\sigma(x_i)$ will increase gradually until G_{sub} will not change anymore. Constraint violation frequency $\sigma(x_i)$ will achieve its maximum value. Because $x_K \in B(x_{goal}, \tau)$, and $x_K \in X_{free}$, the input leading to x_K from x_{K-1} is a collision free input. From the algorithm description, $\sigma(x_{K-i}) < \frac{N^i - 1}{K^i} < 1$. Thus $\exists \zeta > 0$, such that $\forall x \in P_s, (1 - \sigma(x_i)) > \zeta$.

Combining the above two lower bounds together, $\exists \zeta > 0$ and $\exists \kappa > 0$, such that $p(x_i) > \zeta\kappa$. $\square$

Definition 11 *Ball extension of a set, $B(S, r)$:*

The neighborhood of a set, represented by $B(S, r) = \bigcup_{x \in S} B(x, r)$. If S is replaced by a graph G, S is the set of nodes in G.

Definition 12 *Computer precision error, $\varsigma(x)$:*

Computer precision error equals to $\rho(x, x')$, in which x is the actual value and x' is the value of the floating-point number used to represent x.

Theorem 1 *(Resolution Completeness) Suppose the computer precision error is considered and $\exists \eta > 0$ such that $\forall x \in X, \varsigma(x) < \eta$. If there exists a solution π of length K with tolerance $\frac{\tau}{2}$ for a given Δ, then RC-RRT$_1$ and RC-RRT$_2$ using $B(x, \epsilon)$ will find a solution with τ as tolerance if the following conditions are satisfied:*

1. The system transition equation $s(u, x, t, \delta t)$ meets the Lipschitz condition: $\forall u \in U, x_1, x_2 \in X, \rho(s(u, x_1, t, \delta t), s(u, x_2, t, \delta t)) < L_s \rho(x_1, x_2)$, in which L_s is a strictly positive constant.

2. The computer precision error $\eta < \frac{\tau(L_s - 1)}{2({L_s}^{K+1} - 1)}$.

3. The ball region radius $\frac{{L_s}^{K_c+1}-1}{L_s-1}\eta < \epsilon < \frac{\tau(L_s-1)}{2(L_s^K-1)} - \frac{{L_s}^{K_c+1}-1}{L_s^K-1}\eta$*, in which* K_c *is the length of the largest cyclic path in the search graph* G_∞*.*

Proof: For a given Δ, G_∞ is fixed. A Γ planner generates a graph G_{sub} to search for the solution.

Two computation methods can be used. If every parameter is represented algebraically, an exact geometric computation method [98] can be taken and no computation error will exist. RC-RRT$_1$ and RC-RRT$_2$ using $B(x,0)$ generate $G_{sub} \subset G_\infty$. The other computation method uses floating point numbers to represent the parameters resulting in the numerical computation error. Let $\check{G}_{sub}$ represent G_{sub} generated by using the floating point computation. The $\check{G}_{sub}$ generated by RC-RRT$_1$ and RC-RRT$_2$ approximate G_∞ with the computation error. It is possible that the algorithm reports a wrong solution when the computation error leads a path into the goal region even though it is not a solution path. The computation error is related to the computer architecture, algorithm, and state transition equation. It is difficult to analyze all of these computation error factors. To show the effect of the computation error on the completeness, a simple computation error model is used. In this model, we assume only $\varsigma(x)$ is considered, and all other computations are accurate.

Let $U_s = (u_1, u_2, \cdots, u_p)$ denote any input sequence. By applying U_s on a state x at time t_0, we get the final state $\Phi(U_s, x, t_0) = s(u_p, (s(u_{p-1}, \cdots (s(u_1, x, t_0, \delta t)) \cdots, t_0 + (p-2)\delta t, \delta t)), t_0 + (p-1)\delta t, \delta t)$. Using exact computation, because

$$\rho(s(u, x_1, t, \delta t), s(u, x_2, t, \delta t)) < L_s \rho(x_1, x_2),$$

the distance between two final states resulted from x_1 and x_2 will be

$$\rho(\Phi(U_s, x_1, t_0), \Phi(U_s, x_2, t_0)) < {L_s}^q \rho(x_1, x_2).$$

For floating point methods, by only considering the computer precision error in each stage and ignoring all of the other computation errors, the error between the two final states is

$$\rho(\Phi(U_s, x, t_0), \Phi(U_s, x', t_0)) < \frac{{L_s}^{q+1} - 1}{L_s - 1}\eta.$$

Because the complete conditions varies with different G_∞, RC-RRTs, and computation methods, we will provide the completeness arguments for RC-RRTs in the following three parts:

Part I. Accessibility graph with finite N_∞ and no cyclic paths Here, G_∞ is a tree rooted at x_{init}. Both RC-RRT$_1$ and RC-RRT$_2$ can be used in this condition. Only complete conditions of RC-RRT$_1$ are presented in this part because completeness conditions for RC-RRT$_2$ for G_∞ with finite N_∞ and no cyclic paths is same as that for G_∞ with finite N_∞ and cyclic paths, which will be discussed in Part II.

If the exact computation method is used, the completeness can be guaranteed by exploring until $G_{sub} = G_\infty$.

If floating point computation is used, the analysis is shown in Fig. 5.1. The solution path $\pi_u = \{u_1, u_2, \cdots, u_K\}$ is an input sequence of length K and the system transition equation satisfies Lipschitz condition, the initial state precision error $\eta = \rho(x_{init}, x'_{init})$ will lead to the error $\rho(\Phi(\pi_u, x_{init}, t_0), \Phi(\pi_u, x'_{init}, t_0)) < \frac{{L_s}^{K+1}-1}{L_s-1}\eta$ at final state. If $\eta < \frac{\tau(L_s-1)}{2({L_s}^{K+1}-1)}$, the final computation error $\rho(f(P_s, x_{init}), f(P_s, x'_{init})) < \tau$. The RC-RRT will find the solution with tolerance τ if $\exists \pi$ and $\exists x_{goal} \in X_{goal}$ such that $\rho(\pi(t_f), x_{goal}) < \frac{\tau}{2}$.

Part II. Accessibility graph with finite N_∞ and cyclic paths Because only the exploration information is used in RC-RRT$_1$ to exclude repeated states, the repeated states generated by cyclic paths must be included. RC-RRT$_1$ might continue running even though all states are explored. RC-RRT$_2$ is suitable for a G_∞ that has finite N_∞ and cyclic paths.

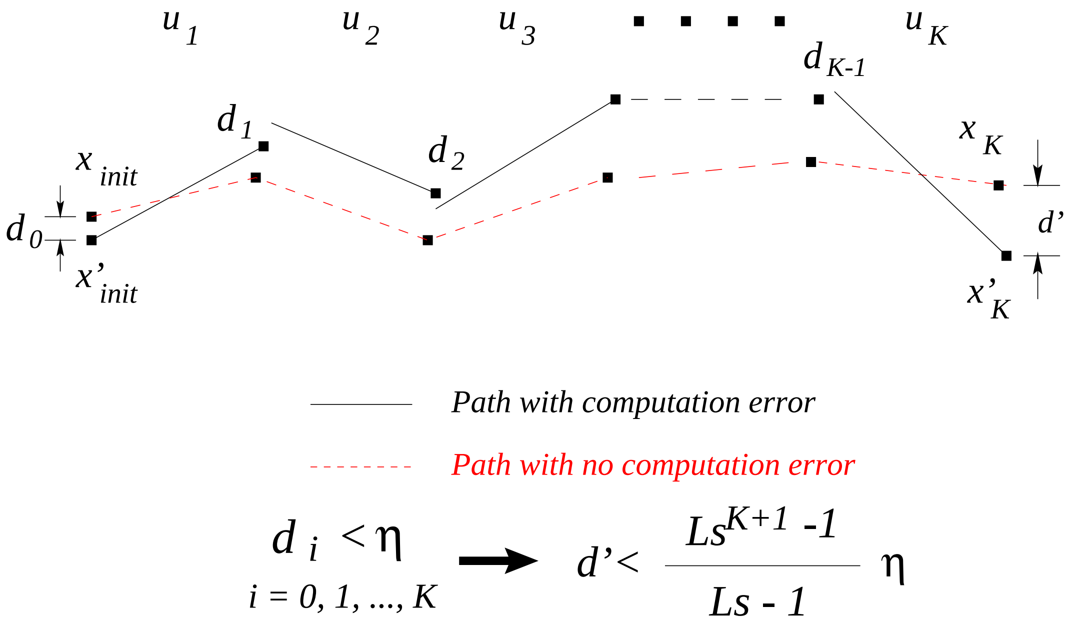


Figure 5.1 Accumulated computation error resulted from computer precision error.

Using exact computation, G_{sub} generated by RC-RRT$_2$ with 0 radius neighborhood is always a subset of G_∞. Because each iteration a new node will be added to G_{sub}, after finite iterations, $G_{sub} = G_\infty$ and the completeness is achieved. If $B(x, \epsilon), \epsilon > 0$ is used to exclude repeated states, G_{sub} might only be an approximation of original G_∞ with some nodes and edges missing. A set of nodes, $S_x \subset N_\infty$ might be replaced by only one state node in N_{sub}. If ϵ is too large, a solution might be missed by exceeding the tolerance τ. For example, let $S_x = \{x_1, x_2, \cdots, x_k\} \subset N_\infty$, and $x' = x_i, x_i \in S_x$ will represent S_x in G_{sub} if x_i is the first state node visited in S_x during the search and $\forall x \in S_x, \rho(x, x_i) < \epsilon$. Assume a solution $\pi_x = (x_0, x_1, x_2, \cdots, x_p, \cdots, x_K)$ exists, in which x_p is the first state in π_x replaced by state, x_i, such that $x_p \in S_x$ and $x_p \neq x_i$. Let $x_K = \Phi(u_p, x_p, t)$, in which $x_p = \pi(t)$ and $u_p = \{u_{p+1}, u_{p+2}, \cdots, u_K\}$ is the input sequence leading x_p to x_K. The error resulting from ϵ in $x_i \in \pi_x$ will be $L_s^{(K-p)}\epsilon$. Under the worst-case when $x_1 \in \pi_x$ is replaced, $\epsilon < \frac{\tau(L_s-1)}{2(L_s^K-1)}$ will guarantee a solution with τ tolerance will be found.

Considering the computation error, RC-RRT$_2$ with $B(x, 0)$ cannot be applied because the two same states resulting from a cyclic path might be different because of the computation error. The $B(x, \epsilon), \epsilon > 0$ has to be used to control the computation error. If ϵ is too small, two same states will still be two different state nodes in N_{sub}. Assume that K_c is the maximum length of all of cyclic paths, if ball radius $\epsilon > \frac{{L_s}^{K_c+1}-1}{L_s-1}\eta$, then two same states will stay in the same ball region for any cyclic path in G_∞. The $\epsilon < \frac{\tau(L_s-1)}{2(L_s^K-1)} - \frac{{L_s}^{K_c+1}-1}{(L_s-1)L_s^{(K-1)}}\eta$ is needed to control the approximate precision.

Part III. Accessibility graph with infinite N_∞ If there exists a Δ locally controllable state in X_{free} for a given Δ, N_∞ is infinite. Only RC-RRT$_2$ using $B(x, \epsilon), \epsilon > 0$ is suitable for this problem. It is easy to prove that RC-RRT$_2$ using $B(x, \epsilon), \epsilon > 0$ can only generate G_{sub} to approximate G_∞ by considering that $\mu(X_{free})$ is a finite constant, and $\exists c > 0$ such that $\forall x \in N_{sub}, \mu(B(x, \epsilon) - B(N_{sub} - \{x\}, \epsilon)) > c$, in which c is a constant and $\mu(S)$ is the Lebesque measure function defined on X.

For exact computation, similar to the above situation, $\epsilon < \frac{\tau(L_s-1)}{2(L_s^K-1)}$ will guarantee the solution is found.

For floating point computation, $\frac{{L_s}^{Kc+1}-1}{L_s-1}\eta < \epsilon < \frac{\tau(L_s-1)}{2(L_s^K-1)} - \frac{{L_s}^{Kc+1}-1}{(L_s-1)L_s^{(K-1)}}\eta$. $\square$

Extension of resolution completeness RC-RRTs use $B(x, \epsilon)$ neighborhood to discretize X. Actually different neighborhoods can be employed. They can be grids, balls, and so on. These neighborhoods can either be disjoint or intersected. Here we will prove that a method using different neighborhoods to discretize X has the same resolution completeness conditions. To make a method be independent of a specific neighborhood, we introduce the following definition.

Definition 13 *Covering of X, $\Pi(X)$:*

A covering of X divides X into n_p sets, $S^1, S^2, \cdots, S^{n_p}$, which satisfies the following conditions:

1. $X = \bigcup_{i=1}^{n_p} S^i$.

2. $\exists \kappa > 0, \forall i \in [1, 2, \cdots, n_p], \mu(S^i - \bigcup_{j \in [1,2,\cdots,i-1,i+1,\cdots,n_p]} S^j) > \kappa$.

If the sets in $\Pi(X)$ are disjoint, $\Pi(X)$ is a partition of X.

In RC-RRT$_2$, $B(x, \epsilon)$ is used to discretize the state space such that G_{sub} with finite N_{sub} is generated for G_∞ with infinite N_∞. After finite number of iterations, no new nodes can be added to G_{sub}. A finite set $S_e = \{x \mid x \in X_{free} - B(G_{sub}, \epsilon)\}$ can be determined such that $X_{free} = B(G_{sub}, \epsilon) \bigcup B(S_e, \epsilon)$. One method to construct S_e is given as follows: Initially, $S_e = \varnothing$; the first state $x_1 \in X_{free} - B(G_{sub}, \epsilon)$ is added into S_e; then the second state $x_2 \in X_{free} - (B(G_{sub}, \epsilon) \bigcup B(S_e, \epsilon))$ is added into S_e; continue this operation to add new state S_e until $X_{free} - (B(G_{sub}, \epsilon) \bigcup B(S_e, \epsilon)) = \varnothing$. After the construction of S_e, we generate a covering $\Pi(X) = \{S \mid S = B(x, \epsilon), x \in N_{sub} \bigcup S_e\}$.

Actually no matter which kind of neighborhood is used to discretize X, a covering corresponding to this neighborhood can always be derived; otherwise, N_{sub} might be infinite. Even though $\Pi(X)$ varies using different neighborhoods, from the proof for Theorem 1, we can easily obtain the following corollary.

Corollary 1 *Let $\mathbb{N}(x)$ be a neighborhood of x used by RC-RRT to discretize the state space, RC-RRT has the same resolution completeness conditions for any $\mathbb{N}(x)$, if:*

1. *$\Pi(X)$ can be constructed from $\mathbb{N}(x)$;*
2. *$\forall S^i \in \Pi(X)$, and $\forall x_1, x_2 \in S^i$, $\rho(x_1, x_2) < \epsilon$.*

The proof of resolution completeness is based on G_∞. Because G_∞ is only related to the problem and has nothing to do with a specific method employed to solve the problem. The resolution completeness can be applied to any Γ methods. We can get the following corollary.

Corollary 2 *For a given Δ, Γ methods have the same resolution complete conditions as RC-RRT.*

5.3 Improvement Over the Original RRT

RC-RRTs are more likely to find π using the same number of iterations for the following reasons.

More likely to choose nodes in π_x RRT-based planners choose x_{near} in the current G_{sub} and extend x_{near} to x_{random}. The solution will be found earlier if the nodes in π_x are more likely to be chosen. Refer to Fig. 4.7; $x_1 \notin \pi_x$ and $x_2 \in \pi_x$. To choose x_2 to extend, x_{random} has to be in $V(x_2)$, which is the Voronoi region of x_2. The probability of this event is $\frac{\mu(V(x_2))}{\mu(X)}$. While in an RC-RRT, if all of the inputs of x_1 are EXPANDED, x_1 is excluded from the state space, a new Voronoi graph will be generated for $N_{sub} - \{x_1\}$.

Let $V'(x_2)$ represent the new Voronoi region of x_2; $\mu(V'(x_2)) \geq \mu(V(x_2))$ such that x_1 is more likely to be chosen as x_{near}.

More likely to choose input in π_u After x_{near} is chosen for extension, $u \in U$ will be selected to generate the most promising node to the solution. The solution will be found earlier if $u \in \pi_u$ is more likely to be chosen. Refer to Fig. 4.8; only the input leading to x_{22} is in π_u. To make this input to be chosen, x_{random} must be in $V(x_{21}) \cap V(x_2)$. In the worst-case, if $V(x_{21}) \cap V(x_2) = \varnothing$, it is impossible to find the solution. For the original RRT, the probability of this event will not increase, even though the input leading to x_{22} has been EXPANDED. While for RC-RRT, when x_{22} is EXPANDED, its corresponding input will be discarded for extending x_2. The volume of the new Voronoi region $V'(x_{21})$ for x_{21} will increase and $\mu(V(x_{21}) \cap V(x_2)) \leq \mu(V'(x_{21}) \cap V(x_2))$. Thus, $u \in \pi_u$ is more likely to be extended.

Completeness under specific conditions From the Section 5.2, RC-RRT-based planners is resolution complete, while the original RRTs can only provide a probabilistic completeness with infinite number of iterations.

From the above description, RC-RRT is more likely to find π than the original RRT using the same number of iterations for problems with poor metric functions.

5.4 Worst-case Analysis

Many path planning methods can be considered as classical AI search technique in the motion planning domain. A systematic search, which keeps the following two tenets, is sometimes used to design path planning algorithms:

1. **Completeness:** The search method will explore every object in the search space. It guarantees that the search method will find the path if one exists.

2. **Efficiency:** The search method will not explore any object in the search space twice. By this restriction, the search method will waste time to explore the space which has been explored before.

Systematic search includes breadth first search, A* search, dynamic programming, and so on. A global optimal solution is normally found. Randomized methods are well known for sacrificing the global optimality for the speed and can often find solutions faster than systematic search methods. RC-RRTs in the thesis combine randomized techniques and systematic search together to return an optimal solution in the explored space in finite number of iterations.

For a Γ method, initially x_{init} is the only node in N_{sub}. In each iteration, a node, n, and edges connecting n and G_{sub} might be added. Eventually after a finite number of iterations, no more new nodes can be added. We use $\bar{G}_{sub}(\bar{N}_{sub}, \bar{E}_{sub})$ to represent the final G_{sub}. Because nodes in N_∞ might be added to G_{sub} in different order, there will be different $\bar{G}_{sub}$.

Lemma 2 *For a given Δ, assume each node in N_∞ costs one unit of space for storing. If the resolution completeness conditions are satisfied, let m be the number of inputs in U, n be number of nodes in $\bar{G}_{sub}$, $n_{max} = \max_{\forall \bar{G}_{sub}} n$, and let n_p be the number of sets in $\Pi(X)$ which corresponds to the chosen neighborhood used in RC-RRT$_2$. An RC-RRT$_2$-based planner needs at most $n_{max}m \leq n_p m$ iterations and $n_{max} \leq n_p$ units of space to determine whether there exists a solution.*

Proof: To bound the number of iterations in the worst-case, we first need to find the upper bound for the number of nodes in $\bar{N}_{sub}$. n_∞ represents the number of nodes in N_∞.

Upper bound for number of nodes in $\bar{N}_{sub}$ For a given Δ, whose G_∞ has a finite N_∞, a $\Pi(X)$ can be constructed, in which $\forall S^i \in \Pi(X)$, if $\exists n_1 \in N_\infty \cap S^i$, then

$\forall n_2 \in N_\infty - \{n_1\}, n_2 \notin S^i$. Thus, $n_\infty \leq n_p$. Because the resolution completeness conditions are satisfied, using this $\Pi(X)$, any $\bar{N}_{sub}$ has the same number of nodes as N_∞. Thus $n_{max} = n_\infty \leq n_p$.

For Δ, whose G_∞ has infinite N_∞, a $\Pi(X)$ is required to meet the resolution completeness conditions. For the $\Pi(X)$, $\exists S^i \in \Pi(X)$ and $\exists N_i = \{n_{i1}, n_{i2}, \cdots, n_{ij} \in N_\infty, j \geq 2\}$ such that $N_i \subset S^i$. According to RC-RRT$_2$ algorithm (Fig. 4.14), only $n_{ik} \in N_i$ will be in $\bar{N}_{sub}$ when n_{ik} is the first node in N_i to be added to G_{sub}. $\forall n \in N_i - \{n_{ik}\}$ will not be in $\bar{N}_{sub}$. Because $\forall \bar{G}_{sub}$ and $\forall S^i \in \Pi(X)$, if $\exists n_1 \in N_\infty \cap S^i$, then $\forall n_2 \in N_\infty - \{n_1\}, n_2 \notin S^i$, $n_{max} \leq n_p$.

Space requirement Because we assume each node costs one unit of space, RC-RRT$_2$ needs at most $n_{max} \leq n_p$ units of space.

Number of iterations According to the termination condition in Subsection 4.5.1, RC-RRT$_2$-based planner will stop searching when all of the inputs for all state nodes existing in N_{sub} are EXPANDED. In the RC-RRT$_2$ algorithm description (Fig. 4.14 and Fig. 4.12), marking one input for a state node as EXPANDED needs at most one iteration. Thus at most $n_{max}m \leq n_p m$ iterations are required to determine whether a solution exists for a given Δ. $\square$

CHAPTER 6 EXPERIMENTS

Our implementation was built on top of the C++ Motion Strategy Library. Experiments were conducted on a PC running Red Hat Linux 6.2 with a single 1000 Mhz Athlon CPU. The first series of experiments are lane changing experiments. Based on different car models, the experiments are used to compare the performance of RC-RRT-based planners and the original RRT planners. The comparison has two parts: the first part is to compare the exploration ability of the RC-RRT and the original RRT. The second part is to compare the performance and reliability of the planners based on original RRTs and RC-RRTs. In the second series of experiments, several challenging problems were solved by using RC-RRT-based planners, which include virtual driving of a 9-dimensional car model and a trajectory design for a 12-dimensional spaceship.

6.1 Lane Changing Experiments

Lane changing problems comes from the automotive industry. They are used to test the performance of the design of new cars. Figure 6.1 shows a specific lane changing problem (referred as Consumer Union Short Course in the automotive industry [6]), in which a car drives at 96 kph and needs to complete a lane changing maneuver in a 305m stretch of road. The performance comparison of the RC-RRT-based planners and the original RRT planners is based on this problem.

Figure 6.1 Lane changing experiment.

6.1.1 Exploration ability comparison

RRT-based planners search for the trajectory directly. The generated search structure will cover more and more of the state space until a solution is found. To focus on comparing the exploration ability of both RRTs, the single goal-biased RRT planner framework (Fig. 4.19) is employed with a weighted Euclidean metric function. The original RRT and the RC-RRT_1 are adapted into the planner to generate results that illustrate the exploration ability of RC-RRTs and RRTs. To display the metric issue in the RRT for systems with differential constraints, a holonomic model and the 5-dimensional nonlinear vehicle dynamic model, introduced in Chapter 3, are used in the experiments. In the holonomic model, states can change in any direction without considering the differential constraints. To illustrate the explored state space, snapshots of the search structure are taken after some number of iterations. Figure 6.2 displays the snapshots of the RRT for the holonomic system. The exploration is fast. Figure 6.3 and Fig. 6.4 show the snapshots of the RRT and the RC-RRT for systems with differential constraints. From Fig. 6.3 and Fig. 6.4, it is clear that because of the differential constraints the original RRT covers much less space. The RC-RRT improves the exploration as shown in Fig. 6.4.

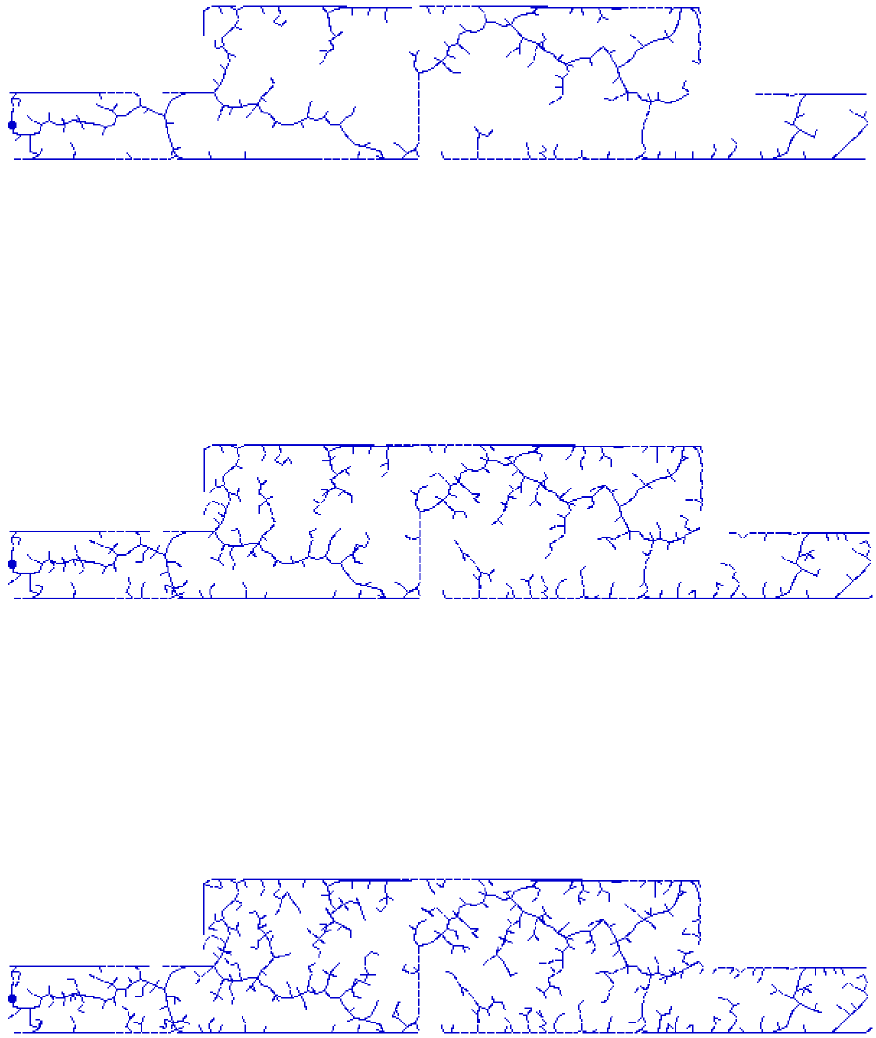


Figure 6.2 The RRT exploration (after 2000, 4000, 6000 iterations) for holonomic model, using a weighted Euclidean metric.

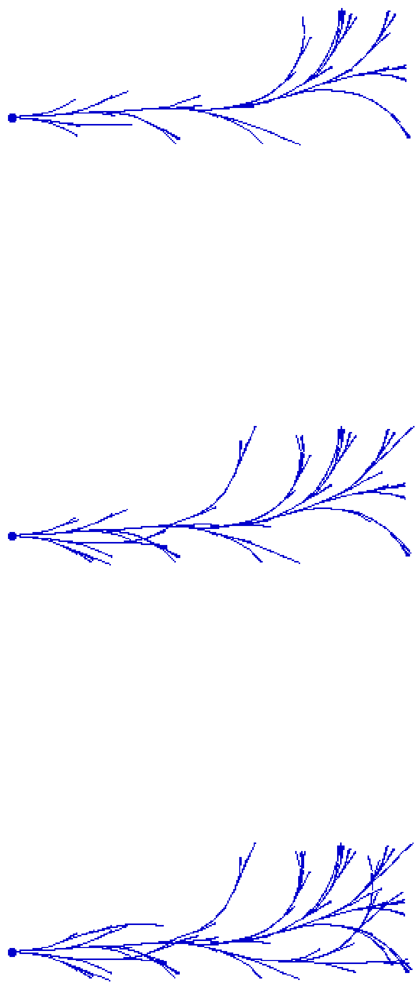


Figure 6.3 The RRT exploration (after 2000, 4000, 6000 iterations) for the car model with differential constraints, using the same metric.

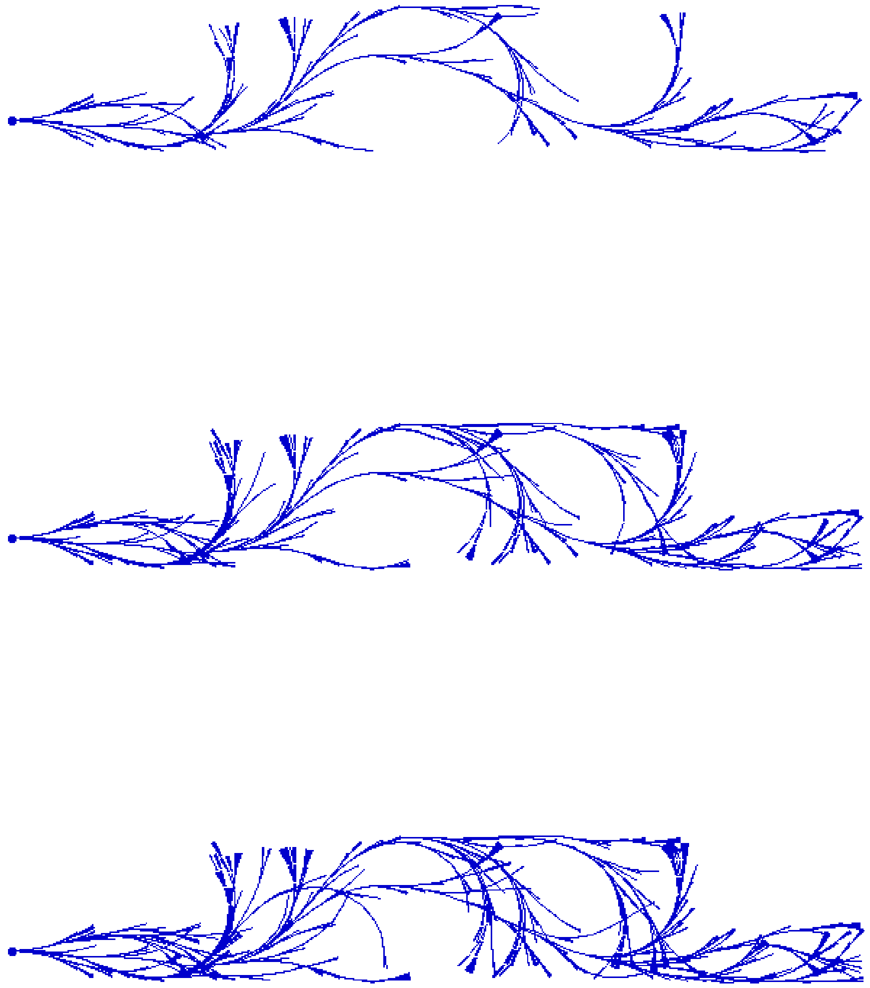


Figure 6.4 The RC-RRT exploration (after 2000, 4000, 6000 iterations) for the car model with differential constraints, using the same metric.

Table 6.1 Comparison of the RRT-based planner and the RC-RRT-based planner.

	RRT Planner			RC-RRT Planner		
Iteration	2000	4000	8000	2000	4000	8000
Num. of Succ.	1	0	4	23	37/50	49
Num. of Nodes	336	-	636.5	1359	2542	3283
Num. of Col. Det.	56.9	-	27.3	2.48	4.49	5.75
Aver. Time	10.61	-	52.15	11.9	28.8	44.7

6.1.2 Planner reliability comparison

To show the improvement in the reliability of the planners, the 5-dimensional car model is employed in lane changing experiments. Planners are based on the bi-directional planner framework (Fig. 4.20). Fifty trials were performed in which six versions were run: the original RRT with 2000, 4000, and 8000 iterations, and RC-RRT with 2000, 4000, and 8000 iterations. Table 6.1 gives some comparative statistics for solutions to the lane changing problem, in which "Iteration" means how many iterations the search tree extends; "Num. of Succ." means the number of successes out of 50; "Num. of Nodes" means the average nodes in the search tree; "Num. of Col. Det." means the average collision detection times (in thousands); "Aver. Time" means the average time needed to find the solution. Note that the success rate improves dramatically using the RC-RRT with the same number of iterations. Furthermore, the average number of nodes generated by the RC-RRT is greatly increased, indicating greater exploration. Also, less collision detection was performed by the RC-RRT. Although computation times are comparable, note that most of the original RRT trials result in failure.

Figure 6.5 A virtual driving experiment.

6.2 Virtual Driving in a Town

Figure 6.5 shows an experiment of driving a car at 72 kph through a virtual town which is 300m by 300m. The nine-dimensional car model is defined in Section 3.2. The model considers the rolling angle and the rolling speed of the car. The pressure on the individual tires varies because of rolling effects. If the pressure on one tire is negative, the car is in a dangerous state. This makes it very difficult to control.

The RRT and RC-RRT were adapted into the single goal-biased RRT planner framework. Each planner was used to solve the virtual driving problem for 50 trials. For each trial, the planner will keep running until a solution is found; however, if the running time for one trial is over 48 hours, the left trials are aborted. The RC-RRT-based planner found the solution in each trial with an average of 1629.62 seconds. For the RRT-based planner, we did fifty trials twice. In the first fifty trial, the first planner failed after 48 hours; in the second fifty trials, the eighth trial failed after 48 hours.

6.3 Trajectory Design for a Floating Spacecraft

The final experiment involves moving an underactuated spacecraft in a 3D grid by firing thrusters (Fig. 6.6). The model is a twelve-dimensional system defined in Section 3.2. The spacecraft can translate and rotate in 3D space. Three thrusters provide the driving forces and torques, which are applied in directions that avoid the mass center (Fig. 3.3). The RRT-based and RC-RRT-based planners are constructed using the bi-directional planner framework. Each planner was used to solve the problem for 50 trials. For each trial, planner will keep running until a solution is reported; however, if a trial runs over 48 hours, the left trials are aborted. The RC-RRT-based planner solved the problem fifty times with an average of 8059.31 seconds. We did the fifty trials on the RRT-based planner twice. They both failed in the first trial after 48 hours.

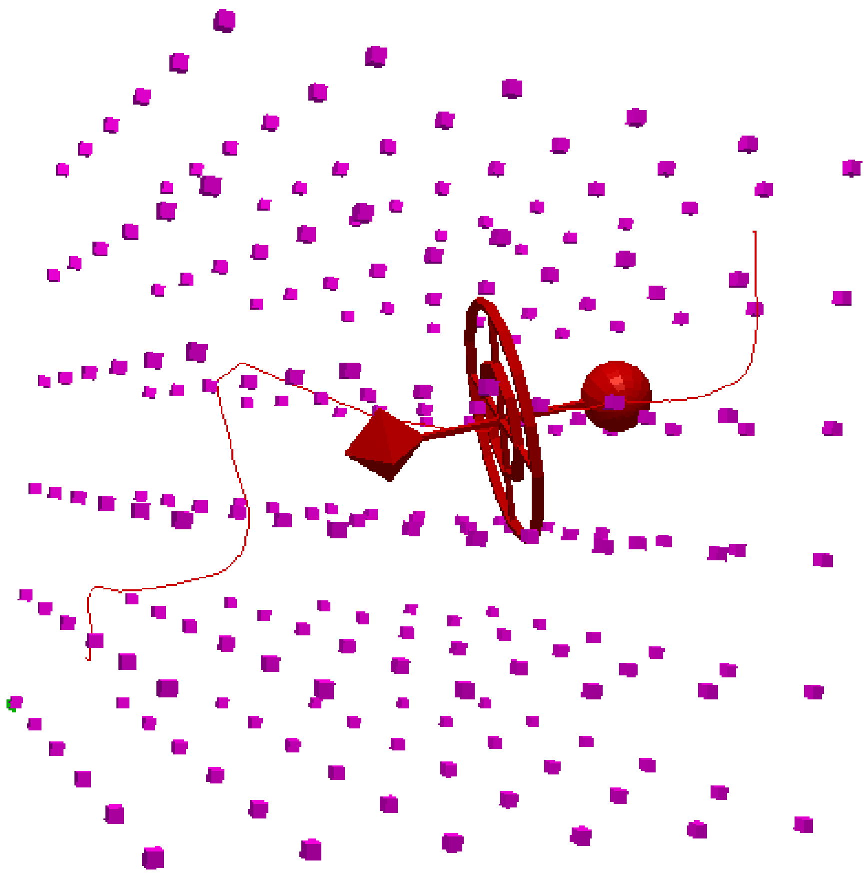


Figure 6.6 A trajectory design experiment for an underactuated spacecraft with three thrusters moving in a 3D grid.

CHAPTER 7 CONCLUSION

Although RRTs are able to overcome typical local minima problems and show good performance for holonomic path planning problems, their performance for difficult problems with differential constraints is not reliable because the efficiency of the search still depends on the quality of the heuristic metric function, which can provide poor guidance information. In this thesis, by combining randomized techniques with systematic search, searching for the trajectory directly, and backtracking to avoid traps, RC-RRT-based planners are presented. These new methods are suitable for problems that involve obstacles, high dimensionality, and nonlinear systems with drift. In particular, sensitivity to poor metrics is reduced by applying information gathered during the search. The exploration information and neighborhoods exclude repeated states from the tree, and make the planner more likely to explore the unexplored state space. The constraint violation frequency information collected during the exploration provides global constraint information which is distributed in the search graph. Our analysis shows that RC-RRTs are more reliable than the original RRTs, and achieve *resolution completeness*. The performance improvement of RC-RRTs over the original RRTs is verified in lane changing experiments. RC-RRT-based planners also solved challenging problems that include virtual driving in a town with a 9-dimensional nonlinear car-like robot and the trajectory design for a 12-dimensional underactuated spacecraft.

BIBLIOGRAPHY

[1] N. M. Amato and Y. Wu. A randomized roadmap method for path and manipulation planning. In *IEEE Int. Conf. Robot. & Autom.*, pages 113–120, 1996.

[2] J. Barraquand and J.-C. Latombe. A Monte-Carlo algorithm for path planning with many degrees of freedom. In *IEEE Int. Conf. Robot. & Autom.*, pages 1712–1717, 1990.

[3] J. Barraquand and J.-C. Latombe. Robot motion planning: A distributed representation approach. *Int. J. Robot. Res.*, 10(6):628–649, December 1991.

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

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

[6] J. Bernard, J. Gruening, and K. Hoffmeister. Evaluation of vehicle/driver performance using genetic algorithms. In *Society of Automotive Enfineers*, number 980227, 1998.

[7] J. Bernard, J. Shannan, and M. Vanderploeg. Vehicle rollover on smooth surfaces. In *SAE Technical Paper Series*, number 891991, Dearborn, Michigan, 1989.

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

[9] J. T. Betts. Survey of numerical methods for trjectory optimization. *AIAA Journal of Guidance, Control, and Dynamics*, 21(2):193–207, 1998.

[10] J. Bobrow, S. Dubowsky, and J. Gibson. Time-optimal control of robotic manipulators. *Int. J. Robot. Res.*, 4(3), 1985.

[11] J. D. Boissonnat and S. Lazard. A polynomial-time algorithm for computing a shortest path of bounded curvature amidst moderate obstacles. In *Proc. ACM Symposium on Computational Geometry*, pages 242–251, 1996.

[12] V. Boor, N. H. Overmars, and A. F. van der Stappen. The gaussian sampling strategy for probabilistic roadmap planners. In *IEEE Int. Conf. Robot. & Autom.*, pages 1018–1023, 1999.

[13] R. A. Brooks. Solving the find-path problem by good representation of free space. *IEEE Trans. Syst., Man, Cybern.*, 13(3):190–197, 1983.

[14] R. A. Brooks and T. Lozano-perez. A subdivision algorithm in configuration space for findpath with rotation. In *ICAI*, pages 799–806, 1983.

[15] J. Canny, A. Rege, and J. Reif. An exact algorithm for kinodynamic planning in the plane. *Discrete and Computational Geometry*, 6:461–484, 1991.

[16] J. Canny and J. Reif. New lower bound techniques for robot motion planning problems. In *Proc. IEEE Conf. on Foundations of Computer Science*, pages 49–60, 1987.

[17] J. F. Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA, 1988.

[18] D. Challou, D. Boley, M. Gini, and V. Kumar. A parallel formulation of informed randomized search for robot motion planning problems. In *IEEE Int. Conf. Robot. & Autom.*, pages 709–714, 1995.

[19] C.-T. Chen. *Linear System Theory and Design.* Holt, Rinehart, and Winston, New York, NY, 1984.

[20] M. Cherif. Kinodynamic motion planning for all-terrain wheeled vehicles. In *IEEE Int. Conf. Robot. & Autom.*, 1999.

[21] C. Connolly, R. Grupen, and K. Souccar. A Hamiltonian framework for kinodynamic planning. In *Proc. of the IEEE International Conf. on Robotics and Automation (ICRA'95)*, Nagoya, Japan, 1995.

[22] C. I. Connolly, J. B. Burns, and R. Weiss. Path planning using laplace's equation. In *IEEE Int. Conf. Robot. & Autom.*, pages 2102–2106, May 1990.

[23] B. Donald and P. Xavier. Provably good approximation algorithms for optimal kinodynamic planning for cartesian robots and open chain manipulators. *Algorithmica*, 14(6):480–530, 1995.

[24] B. R. Donald and P. G. Xavier. Provably good approximation algorithms for optimal kinodynamic planning: Robots with decoupled dynamics bounds. *Algorithmica*, 14(6):443–479, 1995.

[25] B. R. Donald, P. G. Xavier, J. Canny, and J. Reif. Kinodynamic planning. *Journal of the ACM*, 40:1048–66, November 1993.

[26] 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.

[27] P. J. Enright and B. A. Conway. Discrete approximations to optimal trajectories using direct transcription and nonlinear programming. *AIAA Journal of Guidance, Control, and Dynamics*, 15(4):994–1002, 1992.

[28] S. Fortune and G. Wilfong. Planning constrained motion. In *STOCS*, pages 445–459, 1988.

[29] E. Frazzoli, M. A. Dahleh, and E. Feron. Robust hybrid control for autonomous vehicles motion planning. Technical Report LIDS-P-2468, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, 1999.

[30] L. Gouzenes. Stategies for solving collision-free trajectories problems for mobile and manipulator robots. *Int. J. Robot. Res.*, 3(4):51–65, 1984.

[31] C.R. Hargraves and S. W. Paris. Direct trajectory optimzation using nonlinear programming and collocation. *AIAA Journal of Guidance, Control, and Dynamics*, 10(4):338–342, 1987.

[32] G. Heinzinger, P. Jacobs, J. Canny, and B. Paden. Time-optimal trajectories for a robotic manipulator: A provably good approximation algorithm. In *IEEE Int. Conf. Robot. & Autom.*, pages 150–155, Cincinnati, OH, 1990.

[33] R. Hermann and A. J. Krener. Nonlinear controllability and observability. In *IEEE Trans. Autom. Control*, pages 728–740, 1977.

[34] D. Hsu, L. E. Kavraki, J.-C. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. In et al. P. Agarwal, editor, *Robotics: The Algorithmic Perspective*, pages 141–154. A.K. Peters, Wellesley, MA, 1998.

[35] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. *Int. J. Comput. Geom. & Appl.*, 4:495–512, 1999.

[36] Y. K. Hwang and N. Ahuja. Gross motion planning–A survey. *ACM Computing Surveys*, 24(3):219–291, September 1992.

[37] Y. K. Hwang and N. Ahuja. A potential field approach to path planning. *IEEE Trans. Robot. & Autom.*, 8(1):23–32, February 1992.

[38] P. Jacobs and J. Canny. Planning smooth paths for mobile robots. In *IEEE Int. Conf. Robot. & Autom.*, pages 2–7, 1989.

[39] P. Jacobs, J.P. Laumond, and M. Taix. Efficient motion planners for nonholonomic mobile robots. In *IEEE/RSJ Int. Workshop on Intelligent Robots and Systems*, pages 1229–1235, 1991.

[40] H. Kaindl and G. Kainz. Bidirectional heuristic search reconsidered. *JAIR*, pages 283–317, December 1997.

[41] T. Karatas and F. Bullo. Randomized searches and nonlinear programming in trajectory planning. In *IEEE Conference on Decision and Control*, 2001.

[42] L. Kavraki, J.C. Latombe, R. Motwani, and P. Raghavan. Randomized query processing in robot motion planning. In *Proc. ACM SIGACT Symp. on Theory of Computing(STOC)*, pages 353–362, 1995.

[43] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. & Autom.*, 12(4):566–580, June 1996.

[44] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *Int. J. Robot. Res.*, 5(1):90–98, 1986.

[45] R. Kindel, D. Hsu, J.-C. Latombe, and S. Rock. Kinodynamic motion planning amidst moving obstacles. In *IEEE Int. Conf. Robot. & Autom.*, 2000.

[46] D. Koditschek. Robot planning and control via potential functions. In O. Khatib, J. J. Craig, and T. Lozano-Pérez, editors, *The Robotics Review 1.* MIT Press, Cambridge, MA, 1989.

[47] D. E. Koditschek. Exact robot navigation by means of potential functions: Some topological considerations. In *Proc. IEEE Int. Conf. Robot. & Autom.*, pages 1–6, 1987.

[48] G. Laffierriere and H. J. Sussman. *Nonholonomic Motion Planning*, chapter A Differential Geometric Approach to Motion Planning. The Kluwer International Series in Engineering and Comuter Science, Boston, MA, 1992.

[49] F. Lamiraux and J.-P. Laumond. On the expected complexity of random path planning. In *IEEE Int. Conf. Robot. & Autom.*, pages 3306–3311, 1996.

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

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

[52] J.-C. Latombe. A fast path planner for a car-like indoor mobile robot. In *Proc. Am. Assoc. Artif. Intell.*, pages 659–665, 1991.

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

[54] J.-P. Laumond. Feasible trajectories for mobile robots with kinematic and environment constraints. In *ECAI*, 1986.

[55] J.-P. Laumond. Finding collision-free smooth trajectories for a non-holonomic mobile robot. In *Proc. Int. Joint Conf. on Artif. Intell.*, pages 1120–1123, 1987.

[56] J. P. Laumond, editor. *Robot Motion Planning and Control*. Springer-Verlag London Limited, London, 1998.

[57] J. P. Laumond and T. Siméon. Motion planning for a two degrees of freedom mobile robot with towing. Technical Report No.89-148, LAAS/CNRS, Toulouse, 1989.

[58] S. M. LaValle. Numerical computation of optimal navigation functions on a simplicial complex. In P. Agarwal, L. Kavraki, and M. Mason, editors, *Robotics: The Algorithmic Perspective*. A K Peters, Wellesley, MA, 1998.

[59] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. TR 98-11, Computer Science Dept., Iowa State University., Oct. 1998.

[60] S. M. LaValle and J. Kuffner Jr. Randomized kinodynamic planning. In *IEEE Int. Conf. Robot. & Autom.*, 1999.

[61] S. M. LaValle and J. Kuffner Jr. Rapidly-exploring random trees: Progress and prospects. In *2000 Workshop on the Algorithmic Foundations of Robotics*, 2000.

[62] Z. Li and J. F. Canny. Robot motion planning with nonholonomic constraints. Technical report, Electronics Research Laboratory, University of California, February 1989.

[63] Z. Li, J. F. Canny, and S. S. Sastry. On motion planning for dextrous manipulation, part i: The problem formulation. In *IEEE Int. Conf. Robot. & Autom.*, pages 775–780, 1989.

[64] T. Lozano-Pérez. Automatic planning of manipulator transfer movements. *IEEE Trans. Syst., Man, Cybern.*, 11(10):681–698, 1981.

[65] T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Trans. on Comput.*, C-32(2):108–120, 1983.

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

[67] E. Mazer, J. M. Ahuactzin, and P. Bessière. The Ariadne's clew algorithm. *J. Artificial Intell. Res.*, 9:295–316, November 1998.

[68] E. Mazer, G. Talbi, J. M. Ahuactzin, and P. Bessière. The Ariadne's clew algorithm. In *Proc. Int. Conf. of Society of Adaptive Behavior*, Honolulu, 1992.

[69] R. M. Murray and S. Sastry. Nonholonomic motion planning: Steering using sinusoids. *Trans. Automatic Control*, 38(5):700–716, 1993.

[70] C. ÓDúnlaing. Motion planning with inertial constraints. *ALGO*, 2(4):431–475, 1987.

[71] C. O'Dunlaing and C. K. Yap. A retraction method for planning the motion of a disc. *Journal of Algorithms*, 6:104–111, 1982.

[72] S. Quinlan and O. Khatib. Elastic bands: Connecting path planning and control. In *IEEE Int. Conf. Robot. & Autom.*, pages 802–807, 1993.

[73] J. A. Reeds and L. A. Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific J. Math.*, 145(2):367–393, 1990.

[74] J. Reif and H. Wang. Non-uniform discretization approximations for kinodynamic motion planning. In J.-P. Laumond and M. Overmars, editors, *Algorithms for Robotic Motion and Manipulation*, pages 97–112. A K Peters, Wellesley, MA, 1997.

[75] J. H. Reif. Complexity of the mover's problem and generalizations. In *Proc. 20th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 421–427, 1979.

[76] J. H. Reif and M. Sharir. Motion planning in the presence of moving obstacles. In *Proc. of IEEE Symp. on Foundat. of Comp. Sci.*, pages 144–154, 1985.

[77] E. Rimon and D. E. Koditschek. The construction of analytic diffeomorphisms for exact robot navigation on star worlds. In *IEEE Int. Conf. Robot. & Autom.*, pages 21–26, May 1989.

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

[79] G. Sahar and J. M. Hollerbach. Planning of minimum time trajectories for robot arms. *Int. J. Robot. Res.*, 5(3):90–100, 1986.

[80] J. T. Schwartz and M. Sharir. On the piano movers' problem: I. The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers. *Communications on Pure and Applied Mathematics*, 36:345–398, 1983.

[81] J. T. Schwartz and M. Sharir. On the piano movers' problem: II. General techniqies for computing topological properties of algebraic manifolds. *Communications on Pure and Applied Mathematics*, 36:345–398, 1983.

[82] J. T. Schwartz and M. Sharir. On the piano movers' problem: III. Coordinating the motion of several independent bodies. *Int. J. Robot. Res.*, 2(3):97–140, 1983.

[83] J. T. Schwartz and M. Sharir. On the piano movers' problem: V. the case of a rod moving in three-dimensional space amidst polyhedral obstacles. *Communications on Pure and Applied Mathematics*, 37:815–848, 1983.

[84] M. Sharir and E. Ariel-Sheffi. On the piano movers' problem: IV. various decomposable two-dimensional planning problems. *Communications on Pure and Applied Mathematics*, 37:479–493, 1983.

[85] Z. Shiller and S. Dubowsky. Global time-optimal motions of robotic manipulators in the presence of obstacles. In *ICRA*, 1988.

[86] Z. Shiller and S. Dubowsky. On computing time-optimal motions of robotic manipulators in the presence of obstacles. *ICRA*, 7(7), Dec 1991.

[87] K. G. Shin and N. D. McKay. Minimum-time control of robot manipulators with geometric path constraints. *IEEE Trans. Autom. Control*, 30(6):531–541, 1985.

[88] A. M. Shkel and V. J. Lumelsky. Incorporating body dynamics into sensor-based motion planning: The maximum turn strategy. *IEEE Trans. Robot. & Autom.*, 13(6):873–880, December 1997.

[89] P. Souéres and J. P. Laumond. Shortest paths synthesis for a car-like robot. In *IEEE Transactions on Automatic Control*, pages 672–688, 1996.

[90] S. Sundar and Z. Shiller. Optimal obstacle avoidance based on the Hamilton-Jacobi-Bellman equation. *IEEE Trans. Robot. & Autom.*, 13(2):305–310, April 1997.

[91] H. Sussman and G. Tang. Shortest paths for the Reeds-Shepp car: A worked out example of the use of geometric techniques in nonlinear optimal control. Technical Report SYNCON 91-10, Dept. of Mathematics, Rutgers University, 1991.

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

[93] M. Taix. *Planification de Mouvement pour Robot Mobile Non-Holonome*. PhD dissertation, LAAS, Toulouse, France, Jan 1991.

[94] G. J. Toussaint, T. Başar, and F. Bullo. Motion planning for nonlinear underactuated vehicles using hinfinity techniques. Coordinated Science Lab, University of Illinois, September 2000.

[95] S. Udupa. *Collision Detection and Avoidance in Computer Controlled Manipulators*. PhD dissertation, California Institure of Technology, 1977.

[96] O. von Stryk and R. Bulirsch. Direct and indirect method for trajectory optimization. *Annals of Operations Research*, 37(1-4):357–73, 1992.

[97] L. Yang and S. M. LaValle. A framework for planning feedback motion strategies based on a random neighborhood graph. In *IEEE Int. Conf. Robot. & Autom.*, 2000.

[98] C. Yap and T. Dubé. *The exact computation paradigm.* World Scientific Press, 1995.

[99] Y. Yu and K. Gupta. On sensor-based roadmap: A framework for motion planning for a manipulator arm in unknown environments. In *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pages 1919–1924, 1998.

ACKNOWLEDGMENTS

I would first like to thank my advisor, Steven M. LaValle, for the advice, encouragement, and help for many aspects of research and life. It is he who gave me the right research direction which led to this thesis. Tammy LaValle's proofreading for my thesis is also greatly appreciated.

I would like also to thank James Bernard for his advice on the vehicle dynamic models, Andrew Olson for providing the exciting geometric model for the spacecraft experiments. Thanks also go to the members of my Program of Study committee, Yan-Bin Jia and Judy Vance. I would also like to thank the help from friends in our lab, such as Anna Atramentov, Libo Yang, Boris Simov and Josh Bertram. The discussion about the vehicle simulation with Xiaopeng Fang is also appreciated.

Special thanks are given to Melanie Eckhart and Clare Polking for their kind help to my study and life.

A gratitude beyond my expression is sent to my parents and my sisters in China. Without their continuous and strong support, it would have been impossible for me to accomplish all of my study and research.

Finally, I would like to thank my wife, Yanghui Gong, for all her understanding and love during my research and studies. I will always be grateful for her unlimited patience during this long and busy time.