SAMPLING-BASED MOTION PLANNING WITH DIFFERENTIAL CONSTRAINTS

BY

PENG CHENG

B.S., Tsinghua University, 1996
M.E., Tsinghua University, 1999
M.S., Iowa State University, 2001

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2005

Urbana, Illinois

This page will be replaced by the Certificate of Committee Approval form.

SAMPLING-BASED MOTION PLANNING WITH DIFFERENTIAL CONSTRAINTS

Peng Cheng, Ph.D.
Department of Computer Science
University of Illinois at Urbana-Champaign
Steven M. LaValle, Advisor

Since differential constraints which restrict admissible velocities and accelerations of robotic systems are ignored in path planning, solutions for kinodynamic and nonholonomic planning problems from classical methods could be either inexecutable or inefficient. Motion planning with differential constraints (MPD), which directly considers differential constraints, provides a promising direction to calculate reliable and efficient solutions. A large amount of recent efforts have been devoted to various sampling-based MPD algorithms, which iteratively build search graphs using sampled states and controls. This thesis addresses several issues in analysis and design of these algorithms. Firstly, resolution completeness of path planning is extended to MPD and the first quantitative conditions are provided. The analysis is based on the relationship between the reachability graph, which is an intrinsic graph representation of a given problem, and the search graph, which is built by the algorithm. Because of sampling and other complications, there exist mismatches between these two graphs. If a solution exists in the reachability graph, resolution complete algorithms must construct a solution path encoding the solution or its approximation in the search graph in finite time. Secondly, planners are improved with symmetry-based gap reduction algorithms to solve their gap problem, which dramatically increases time to return a high quality solution trajectory whose final state is in a small neighborhood of a goal state. The improved planners quickly obtain high quality solutions by minimizing gaps in solution path candidates, which is greatly accelerated using symmetries of robotic systems to avoid numerical integration. Finally, a heuristic is designed to solve metric sensitivity of RRT-based planners, which means that RRT-based meth-

ods have difficulties in escaping local minima when the given metric provides a poor approximation of the cost-to-go. Instead of designing a metric, the heuristic is obtained by collecting collision information online and assigning a real value to each node in the search graph. A node with a higher value means that the number of trajectories from the node that have been detected in collision is larger. Local minima are more likely to be avoided when nodes with smaller values are given higher probability to be extended.

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 was he who guided me into the fascinating world of robotics, kept me focused on the research, revived my interests in mathematics, helped me to improve my writing and presentation, and much more than I can list with the limited space.

I would like also to thank Emilio Frazzoli for his kind help and advice. He is another advisor in my mind. I really enjoyed many insightful and fruitful discussions with him. Thanks also go to Francesco Bullo, Jeff Erickson, Jean-Paul Laumond, Judy Vance, Kevin Lynch, Seth Hutchinson, Vijay Kumar, and Yan-Bin Jia, who gave me many help and suggestions to my research and this thesis writing.

I would also like to thank the help from members of Motion Strategy Laboratory at Iowa State University and University of Illinois, such as Anna Yershova, Benjamin Tovar, Boris Simov, George Zaimes, Hamid Reza Chitsaz, Jason O'Kane, Libo Yang, and Steve Lindemann. I would like to thank Sachin Chitta, Zuojun Shen for pleasant collaborations.

A gratitude beyond my expression is sent to my parents and sisters in China. Without their continuous and strong spiritual and financial 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.

This research was supported by NSF CAREER 0133869 (Frazzoli), NSF CAREER 9875304 (LaValle), NSF 0208891 (Frazzoli and LaValle), and NSF 0118146 (Bullo and LaValle). Any opinions, findings, and conclusions in this thesis are those of the author and do not necessarily reflect the views of the National Science Foundation.

Contents

List of Tables

List of Figures

List of Symbols

$\mathcal{P}$	a motion planning problem with differential constraints
f	the motion equation of $\mathcal{P}$
$\tilde{f}$	the discrete motion equation
U	the input space of $\mathcal{P}$
$\mathcal{U}$	the control space of $\mathcal{P}$
$\bar{\mathcal{U}}$	the set of continuous controls which generate all controls in $\mathcal{U}$
$\tilde{\mathcal{U}}$	the smallest expanded control set which include all controls that could be designed by a local planner
$\tilde{\mathcal{U}}_s$	the sampled control set
u	an input in U
$\tilde{u}$	a control in $\mathcal{U}$
$\mathcal{G}\langle\mathcal{N},\mathcal{E}\rangle$	the reachability graph
$G\langle N,E\rangle$	the search graph
τ	a trajectory in $\mathcal{G}$
τ_ξ	an η-neighboring trajectory in $\mathcal{G}$
$\hat{\tau}_\xi$	the appearance of τ_ξ in G
$\mathcal{D}$	the range of duration of controls in $\bar{\mathcal{U}}$
w	the clearance of a trajectory
ϵ_p	the approximation tolerance
ϵ_s	the tolerance of a solution
ϵ_l	the tolerance for the approximate local planner

ϵ_i	an upper bound on the numerical integration errors
ϵ_n	an upper bound on the arithmetic precision errors
ϵ_d	the dispersion of the state space sampling
ϵ_u, ϵ_t	the dispersions of the control space sampling for nonconnecting local planners
ϵ_x	maximal size of neighborhood where local Lipschitz condition of a local planner holds
ϵ_v	the supremum of duration of controls in $\tilde{\mathcal{U}}$
D_f	an upper bound on the norm of the value of the motion equation
D_p	an upper bound on the absolute value of the first order derivative of coordinates of the control function
d_{inf}	the infimum of $\|x - \tilde{f}(x, \tilde{u})\|$ for all $x \in X$ and $\tilde{u} \in \tilde{\mathcal{U}}$
L_c	Lipschitz constant for the motion equation f
L_d	Lipschitz constant for the discrete motion equation $\tilde{f}$
L_u	Lipschitz constant for the local planner
$\|\cdot\|_\infty$	an infinity norm defined on $\mathcal{U}$

Chapter 1

Introduction

In this chapter, the motion planning problems are first shown in the big picture of robotics, and then the emergence of sampling-based motion planning with differential constraints as a promising method for the motion planning problem is described. Finally, the contributions and organization of the thesis will be presented.

1.1 Motion Planning as an Important Part of Robotics

One of the ultimate objectives of robotics is to design a robotic system which could replace people to autonomously complete laborious, dangerous, and tedious tasks, such as welding, assembly, spray painting, exploring unknown environments, housekeeping,

The Mars Rover Roomba

Figure 1.1: Examples of robotic systems

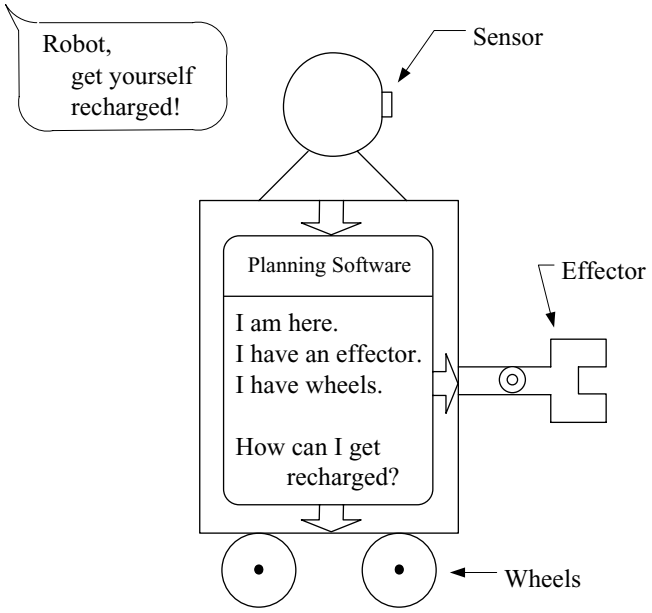


Figure 1.2: The function of the planning software of a robot

recharging, and rescue. Two of the most successful applications of robotic systems are the Mars Rover[1] and Roomba[2] shown in Fig. 1.1. The Mars Rover landed on and is still exploring the surface of the Mars and the Roomba robot is a commercial product that can automatically clean the floor and recharge itself. These robotic systems have the abilities of sensing, planning and action. Sensing processes raw data from sensors to obtain information about the environment and robot, and action affects the state of the environment and robot by applying inputs on the action devices. Planning is achieved by a planning software whose function is to convert sensing information into inputs for the robots. Through the interactions of sensing, planning, and action, the robot could complete assigned tasks. The function of the planning software can be seen in Fig. 1.2, in which the robot is given a task "get yourself recharged" when it is cleaning the floor of the living room as shown in Fig. 1.3, the input to the planning software is the sensing output which tells information of the robot, the output of the software is the inputs that will move the effector and wheels to complete the task.

[1]http://marsrovers.jpl.nasa.gov/home/

[2]http://www.irobot.com/home.cfm

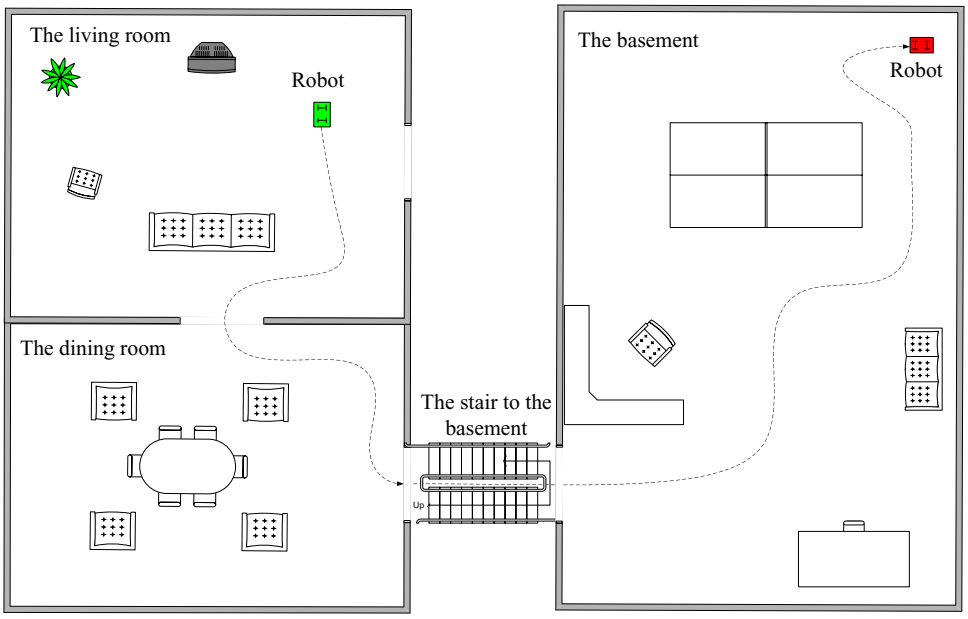


Figure 1.3: A recharging task for the robot

To complete a given task, the planning software needs a characterization of a *work environment* and a robotic system. The work environment is normally a 3D space in which the robot stays for the task. A work environment could include boundaries and objects inside, which are described with geometric primitives, such as straight lines, curves, or planes. In Fig. 1.3, the robot needs to move from the living room to the recharging place at the basement and recharge itself, and therefore the work environment includes the living room, dining room, stairs, basement, and furniture inside. The characterization of the robotic system includes its geometry, sensing model, and action model. The geometry model is described by geometric primitives and a rigid transformation, which together determine the space occupancy of the system at any configuration. The action model shows the relationship between the inputs and the evolution of the state of the system. A state of the system includes its configuration and velocity. If interactions between the robot and work environment are allowed to change the states of the robot and environment, such as collision between them, the action model should also include the relationship between the inputs and the evolution of the state of the environment. In this thesis, however, it is

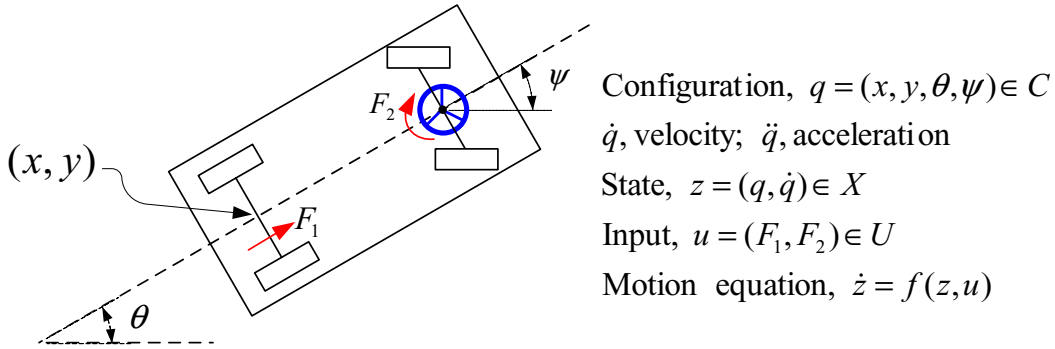


Figure 1.4: The action model (i.e. motion equation in this thesis) for a car-like robotic system

assumed that such interactions do not exist. Under this assumption, the action model is also called *motion equation* of the system. The sensing model shows the relationship between the sensing information and the state of the system and environment.

Assuming that the robot in Fig. 1.2 can only move its wheels, it moves like a car and its action model is shown in Fig. 1.4. Configuration q is represented by configuration variables x, y, θ, and ψ, which are respectively the position, orientation, and steering angle. The set C includes all possible configurations, called the configuration space. A state z includes configuration variables and their first-order time derivatives, i.e, velocity. State space X includes all possible states. Two inputs F_1 and F_2 respectively determine acceleration of forward velocity and velocity of the steering angle. Input space U includes all possible inputs for the system. Motion equation f is represented as a set of Ordinary Differential Equations (ODEs). If the robot is equipped with a Global Positioning System (GPS), then the sensing model will be an identity map assuming that GPS disturbances do not exist, i.e., sensing information from the GPS provides exactly the state of the robot. However, if there exist GPS disturbances, sensing information will not equal the state.

The general tasks for the planning software of robots are normally quite formidable. The first reason is that geometry of the work environment and robot could be complicated and consist of thousands of geometric primitives. The second reason is that the

action model could be under severe constraints so that there is no simple way to apply inputs on the system to achieve desired motions. The action model could be under equality and inequality constraints on configuration variables and their time derivatives. The equality constraints on the configuration variables are called *holonomic constraints*, which normally exist in a robotic system with multiple connected bodies. The inequality constraints on the configuration variables are derived from system design, such as the joint limits. Constraints on the time derivatives of configuration variables are called *differential constraints*, which exist for virtually all robotics systems. For example, since a car cannot go sideways, which is a differential constraint on the action model, it has to go zigzag to parallel park. The third reason is that uncertainties could exist in sensing and action models. With sensing uncertainty, it is difficult for the robot to directly know its state from sensing information. While with action uncertainty, the robot might not go where it will go under no action uncertainty.

To decompose the complexity of the tasks, the planning software is normally implemented in a hierarchical way[3]. Uncertainties in the sensing model are normally handled at the higher level, in which the given task is first decomposed into a sequence of logically related subtasks. "Logically related" means that these subtasks are not independent. For example, one subtask must start after the completion of another subtask, or one subtask must start during the execution of another task when some condition is satisfied. Each subtask is called a motion planning problem which consists of a representation of the robotic system, work environment, initial state, and goal state. At the middle level, an algorithm solves the motion planning problem. If the problem is successfully solved, an open-loop input function will be returned. An input function, which will be called a *control*, is a piecewise-continuous vector-valued

[3]There could exist other hierarchical ways. However, only one of them is used in this thesis to roughly show where the motion planning problem is located in the design of planning software.

function from a time interval to an input space. *"Open-loop"* means that there will be no uncertainty in the action model when applying the control on the system. Applying the control, the characterized system will move from the initial state to the goal state while satisfying all constraints. The resulting state history is called the *trajectory* of the control. At the lower level, actual controls are developed to allow the system to track the trajectory under uncertainties.

The overall planning process for the task in Fig. 1.3 is shown in Fig. 1.5, in which each big hollow arrow represents an algorithm, and results of each algorithm are shown under respective arrows. The task is firstly divided into the following subtasks at higher level, which are executed one after another.

1. move from the current location to the door to the basement;
2. move from the top of the stair to the bottom;
3. move from the bottom of the stair to the recharging place;
4. move the effector to reach the power plug;
5. move the effector to put the power plug to the power outlet.

The motion planning problem of the first subtask is shown in Fig. 1.6, in which the work environment only includes walls, furniture, plants, and electronics of the living and dining rooms, the robot in Fig. 1.2 is considered as a car-like robot since the effector will not move in the subtask. The initial state is in the living room, the goal state is in front of the door to the basement, and the dashed line represents a solution trajectory. It can be seen that the subtask is easier than the original task because it has a smaller work environment, a simpler motion equation, and no uncertainties. Secondly, each of these subtasks is solved by motion planning to obtain an open-loop control for the effector and wheels, which will make the robot complete the subtask when no disturbance exists. In the final stage, the actual control is calculated with

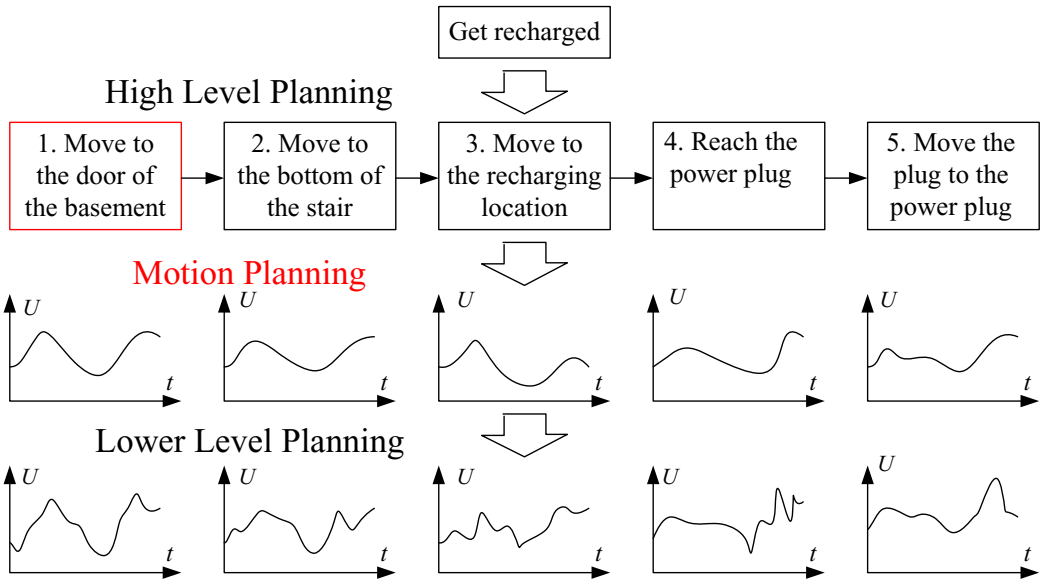


Figure 1.5: A sketch of a hierarchical planning for the task in Fig. 1.3

the low level planning and the open-loop control to allow the robot to complete the task even when the action uncertainty exists. The low level planning is normally achieved by feed-back control laws.

1.2 The Emergence of Sampling-Based Motion Planning with Differential Constraints

With hierarchical planning, no uncertainties are involved in the motion planning problem[4] since they are respectively handled by the high and low level planning. However, the motion planning problems might still include a complicated work environment and action model under severe constraints. Classical two-stage motion planning methods further decompose complexity of motion planning problems and solve them in two steps [1–5]. In the first stage, the motion planning problem is reduced into a *path planning problem* or *Piano Movers' Problem* [6] by considering only geometries of the

[4]In some variation of motion planning problems, uncertainties are considered. However, in this thesis, only the basic motion planning problem is considered, in which uncertainties do not exist.

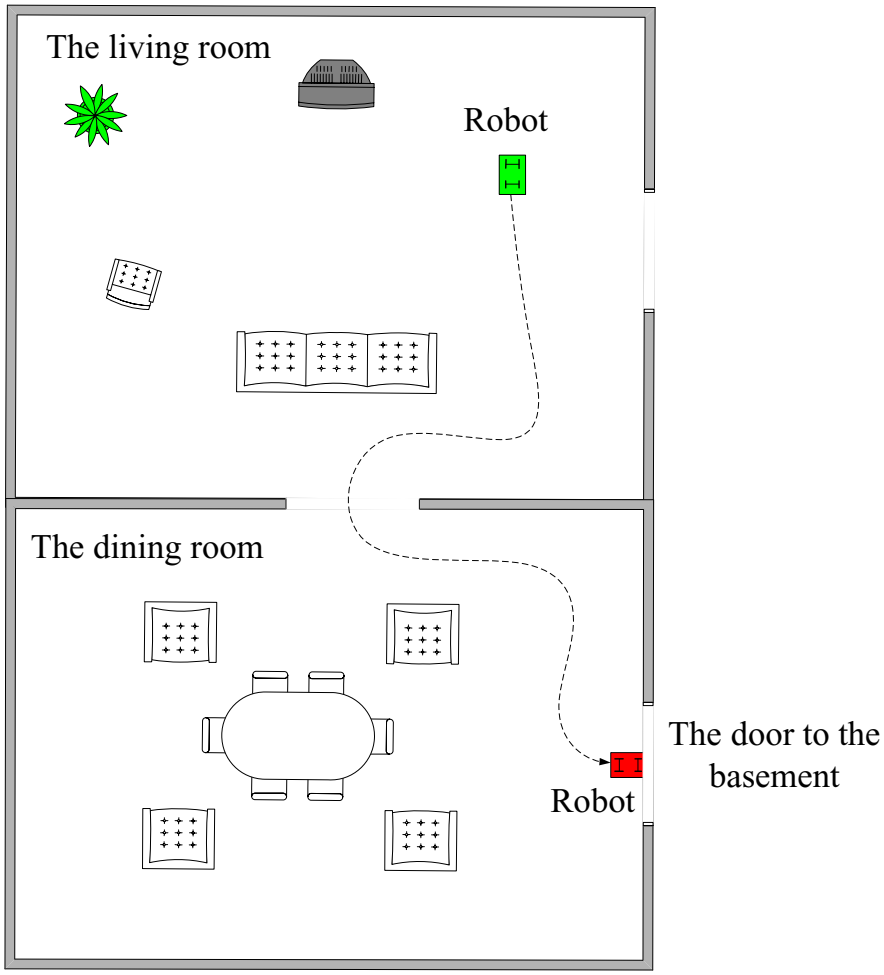


Figure 1.6: The motion planning problem of the first subtask

work environment and robot, whose solution is a collision-free path from the initial configuration to the goal configuration. In the second stage, differential constraints on the action model are considered to time-parameterize the path from the first step into a trajectory.

Path planning problems appeared around 1970 during the study for experimental mobile robots and manipulators [7]. An example of the path planning problem is shown in Fig. 1.7. The piano is considered as a mobile robot that can move freely in any direction on the ground, the work environment includes the room and objects inside that are simplified as rectangular boxes, and the curve is a solution following which the piano could move from the initial configuration to the goal configuration

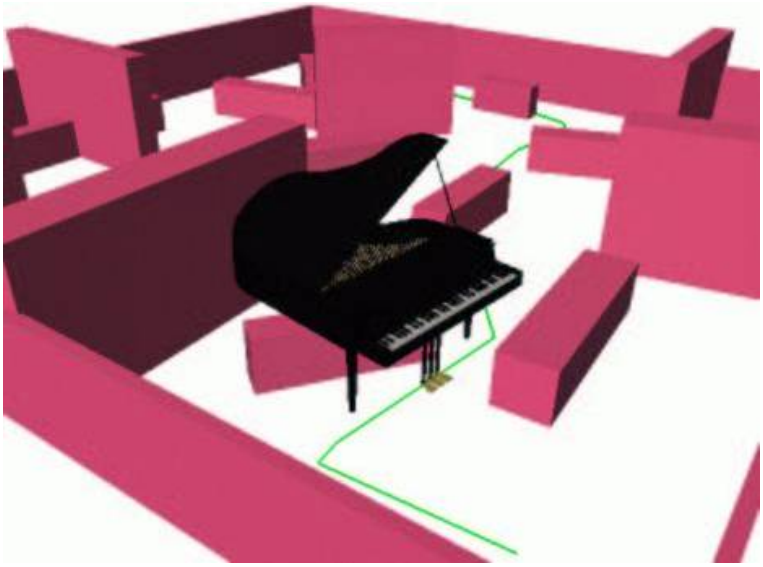


Figure 1.7: An example of the Piano Movers' Problem

without colliding the rectangular boxes. By the introduction of the concept of configuration space [8], the robotic system is reduced into a point, the work environment is mapped into a set that the point cannot enter, and the path planning problem is transformed into a pure geometry problem. The solution to the problem is a continuous path in the configuration space from the initial configuration to the goal configuration that does not intersect the set corresponding to the work environment. The Piano Movers' Problem was shown to be PSPACE-hard [9], in which geometries of the robot and the work environment are characterized by a finite collection of plane primitives. For general Piano Movers' Problem in which geometric primitives in the configuration space are polynomials, two complete algorithms were provided based on cylindrical algebraic decomposition [10] and roadmap [11], whose upper bounds on running time are respectively doubly- and singly- exponential in the dimension of the configuration space. Since complete algorithms for path planning problems take a large amount of time, sampling-based techniques [12–17] have been extensively used to provide a practical solution by sacrificing completeness, in which a set of sampling points are used to represent the configuration space and construct solutions. Refer

to [18; 19] for a complete review of methods for the Piano Movers' Problem.

Classical planning methods have been successfully applied in many areas. In Fig. 1.8, the methods have been used to automatically design the process to assemble the front panel of a car in a collaboration between Fraunhofer-Chalmers Research Centre for Industrial Mathematics and Volvo automobile company. In Fig. 1.9[5], the small molecule in the center is the drug molecule (also called ligand) and the big molecule is the human protein. The ligand will generate the desired effects when it is docked on a specific location on the human protein. Path planning algorithms have been used in rational drug design to verify whether a drug ligand could be docked into the specific location of the protein.

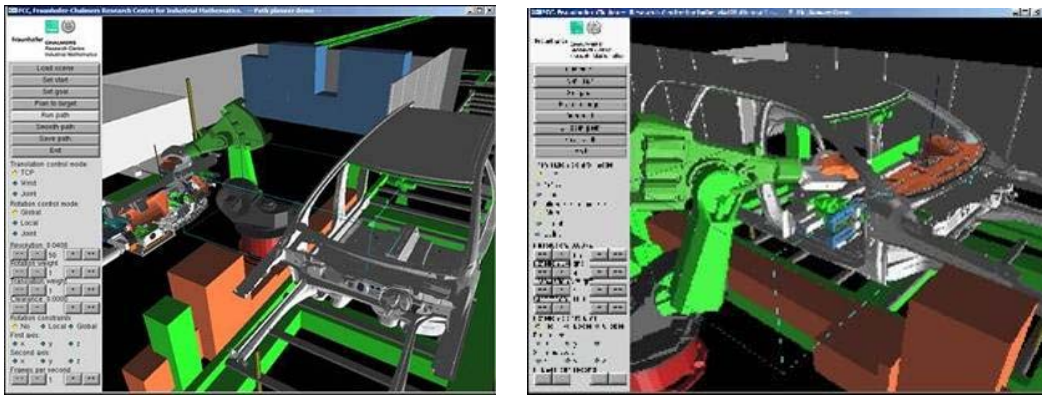
Even though classical two-stage methods have obtain tremendous success, these two-stage methods might suffer the following problems:

1. The path from the first stage might not be transformable into an executable trajectory. An example is shown in Fig. 1.10, in which the straight line is a collision-free path returned from path planning and the car cannot follow the path to complete the parallel parking.

2. The cost associated with the final trajectory could be expensive. The cost could be the time duration or consumed energy. An example is shown in Fig. 1.11, in which the path with smoother turning curves takes the car less time to follow. However, path planning algorithms are more likely to return the path with sharper turning curves.

The direct reason for these problems is that two-stage methods ignore differential constraints of the action model in the first stage. For Fig. 1.10, the car cannot follow the straight path because it cannot go sideways, which could be formulated as

[5]The picture is from http://www.cs.rice.edu/CS/Robotics/bioinformatics/drug.html.



The initial configuration The goal configuration

Figure 1.8: Using path planning to design automated assembly process

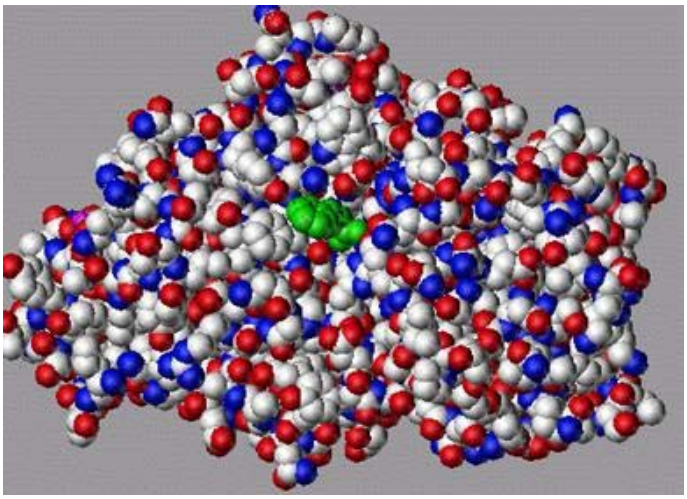


Figure 1.9: Application of path planning in rational drug design

equality constraints in Fig. 1.12:

$$v_s = \dot{x}\sin\theta - \dot{y}\cos\theta = 0, \tag{1.1}$$

in which "$\cdot$" above variables x and y is a shorthand notation for the time derivative operator $\frac{d}{dt}$. For Fig. 1.11, the smoother trajectory takes less time because the car does not need to move slowly at sharp corners, which could be formulated as inequality constraints in Fig. 1.12:

$$\|\dot{v}_f\| = \|[\ddot{x}, \ddot{y}]^T\| < 10. \tag{1.2}$$

Differential constraints considered in robotics are usually *first-order*, or *second-order* because most robot systems are second-order mechanical systems. First-order

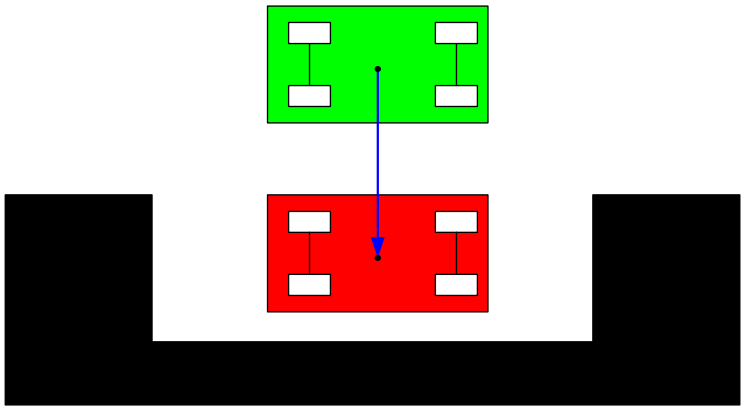


Figure 1.10: An example of the inexecutable path for a car

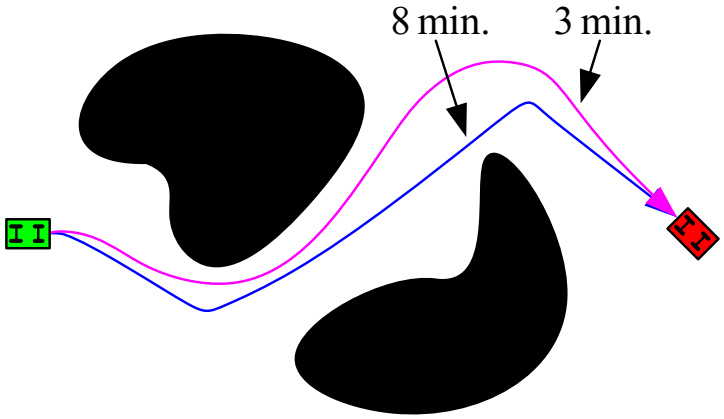


Figure 1.11: An example of the inefficient trajectory for a car

constraints include non-integrable equality constraints, called *first-order nonholonomic constraints*, and inequality constraints over the first-order time derivatives of configuration variables. The equality in Eq. (1.1) and inequality

$$|v_f| = |\dot{x}\cos\theta + \dot{y}\sin\theta| < 50 \tag{1.3}$$

in Fig. 1.12 are examples of first-order nonholonomic and inequality constraints, respectively. Second-order constraints include non-integrable equality constraints, called *second-order nonholonomic constraints* [20], and inequality constraints over second-order time derivatives of configuration variables. The second-order nonholonomic constraints normally come from underactuated systems for which the number of inputs is less than the dimension of the space of admissible velocities. The inequality

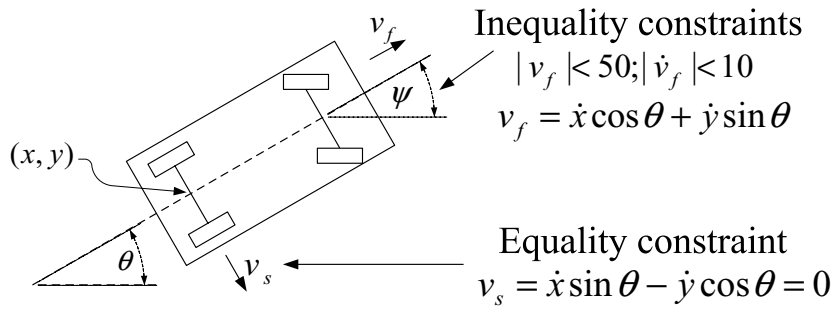


Figure 1.12: Differential constraints on a car system

in Eq. (1.2) is an example of second-order inequality constraints.

Differential constraints exist in the action model of almost every practical motion planning problems. They originate from the following three sources:

1. **Physical laws:** According to the Newton's second law, force is the product of mass and acceleration. Therefore, there exist bounds on accelerations for practical robotic systems since magnitude of forces on these systems is always bounded. The differential constraint in Eq. (1.2) comes from this source since the forces to provide the turning acceleration of the car are the static frictions between the wheels and ground and the static friction only has finite magnitude in reality.

2. **Robotic design:** Many robotic systems are designed to have differential constraints such that the building cost could be cheaper and the system could be easier to control. The first-order constraint in Eq. (1.1) is of this type since the wheels of the car systems are designed to move backward or forward, but not sideways. Second-order nonholonomic constraints are induced in many underactuated systems for the same reason.

3. **Task requirements:** In the application of many robotic systems, there exists differential constraints. For example, if the task of a car-like robot is to transport products on the highway, then it is natural to require that the speed of the system be under the speed limit.

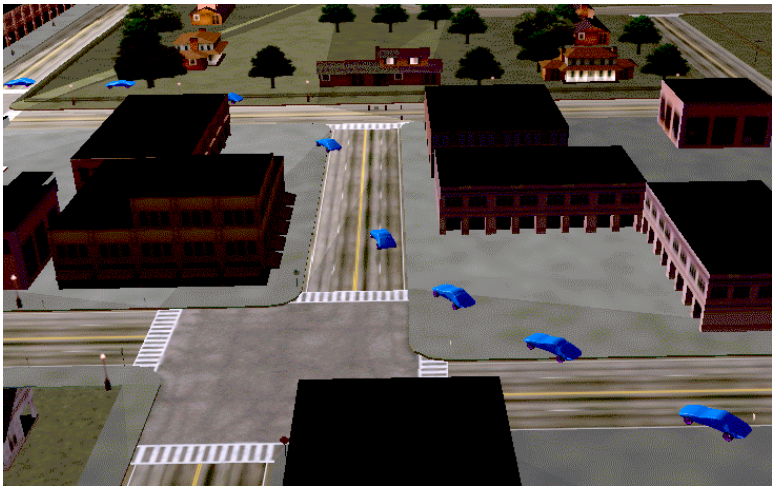


Figure 1.13: Steering a car through a 300m. × 300m. virtual town at 72 kph

To overcome the problems of two-stage methods, many recent planning methods solve the motion planning problem by directly considering differential constraints in the planning process. These methods are called *motion planning with differential constraints*. Since differential constraints are considered in the planning process, their solutions are more reliable and efficient. Specifically, if only first-order nonholonomic constraints are considered, it is generally called *nonholonomic planning* [21] in robotics literature. If second-order constraints are considered, it is called *kinodynamic planning* in [22]. Although in [22] only fully actuated holonomic systems are considered, kinodynamic planning can also take into account underactuated systems. With these new techniques, problems in Fig. 1.13, in which the car is modeled as a nine-dimensional nonlinear dynamical system that accounts for tire loading, skidding, basic suspension effects, and its input is the steering angle, and in Fig. 1.14[6], in which the spacecraft is equipped with three thrusters and modeled as a 12-dimensional dynamics system, can be solved. Both models are described in details in Section 6.3.1.

Since differential constraints are considered in the action model of the kinodynamic planning and nonholonomic planning problems, these problems are believed to be more difficult than ordinary path planning problems. No lower bounds on the

[6]Geometry of the spacecraft and work environment was provided by Andrew Olsen.

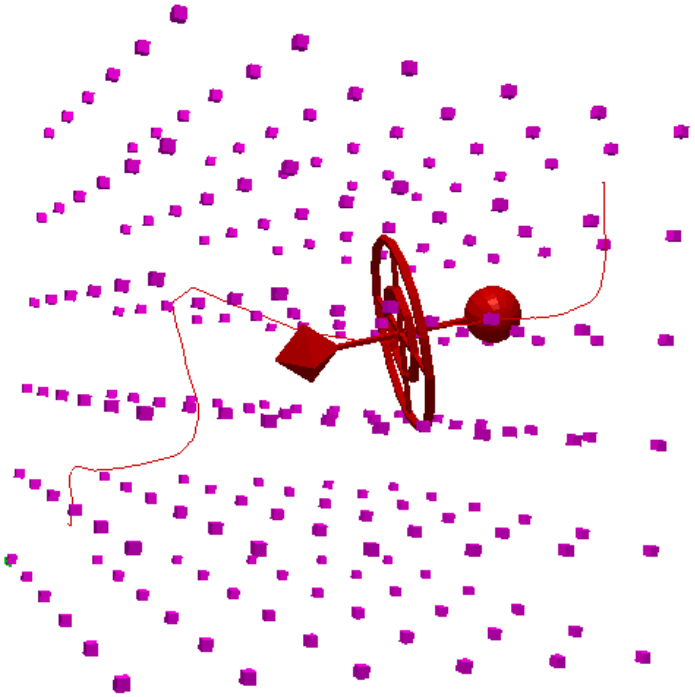


Figure 1.14: Firing three thrusters to move a spacecraft from one corner of a 3D grid to another corner

complexity of the problems have been derived. Also, very few algorithms have been designed to compute exact solutions for general problems that consider the differential constraints. There are only algorithms that compute the exact solutions for some specific low-dimensional problems. Finding an exact time-optimal trajectory with bounds on acceleration and velocity for a point mass moving in an environment with 3D polyhedral obstacles has been proved to be NP-hard [23]. Exact solutions to kinodynamic motion planning problems only exist from point masses with ℓ_∞ bounds on velocity and acceleration in one-dimension [24] and two-dimensions [25].

Steering problems, which are obtained by ignoring the work environments in nonholonomic planning problems, have been actively studied in the last two decades. Analytical solutions have been designed for many robotic systems, which include the Dubins' car [26], Reeds-Sheep Car [27], differential drive [28], differentially flat systems [29], kinematically controllable system [30], nilpotent systems [31], and chained-

form systems [32]. Specifically, optimal solutions even exist for steering problems for the Dubins' car [26], Reeds-Sheep Car [27], and differential drive [28]. However, for general steering problems, numerical methods are used, in which controls are normally approximated by parameterized curves. Nonlinear optimization techniques are then used to optimize these parameters to find suboptimal solutions. However, because the performance of these numerical techniques depends very much on initial guesses of the solutions and has slow convergence on nonlinear problems, this method is only practical for limited cases [33; 34].

Considering the work environments, most of current algorithms use sampling-based techniques [13; 22; 35–42], in which a search graph is incrementally built to construct solutions with sampled controls and states. The search graph is a directed graph, whose nodes are associated with states and whose edges are associated with sampled controls and their trajectories. The control of a path in the search graph is constructed by sequentially concatenating the sampled controls of the edges in the path. The concatenation of two controls is to append one control to the end of the other one by offsetting the time interval of one control by the duration of the other control. Initially, the search graph could include one, two, or more than two nodes. These initial nodes are associated with the initial state, the goal state, or other states. In each iteration, a node n_{cur} in the search graph is firstly chosen by a node selection algorithm. Node selection could use systematic search, such as breadth first search [22; 43], Dijkstra's algorithm [35; 37], or non-systematic search, such as randomized search [13; 37; 38; 44–46]. Then a local planner uses a sampled control $\tilde{u}_{new}$ to extend the state of n_{cur} to a new state while considering differential constraints. Local planners could sample a piecewise constant acceleration [22; 43], a piecewise-constant control [13; 47], and non-constant controls [37; 38; 45; 48–50] which could drive systems between two states either in the state space [45], or in a subspace [37; 38; 48; 49]. The search graph is updated with a given updating policy. If

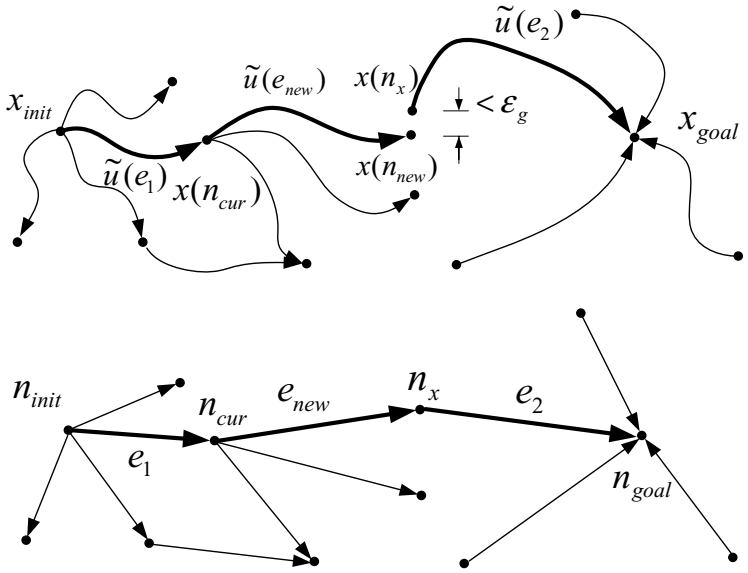


Figure 1.15: An example of the search graph and solution path

a solution checking policy finds a *solution path*, which encodes a solution, the control of the path is returned; otherwise, the algorithm goes to the next iteration until a given termination condition is satisfied. An example of the search graph and solution path is in Fig. 1.15, in which the search graph built from a bi-directional method is in the lower picture, the upper picture shows the states and trajectories associated with nodes and edges in the search graph, the thick lines in the search graph show the solution path, the thick curves in the upper picture are trajectories of the solution path.

1.3 Contributions of this Thesis

While motion planning with differential constraints provides a promising direction, it also generates many interesting and challenging problems, such as motion representation and characterization, lower bounds and exact algorithms for general motion planning problems, sampling-based algorithm design and analysis, and its application. The contributions of this thesis are mainly about analysis and design of sampling-based motion planning with differential constraints.

The first and main contribution is resolution completeness of sampling-based MPD algorithms. For the Piano Movers' Problem, sampling-based methods incrementally build search graphs by sampling configuration space. Latombe provided resolution completeness concept for these planners [18], that is, if there exists a solution, a resolution complete algorithm must find one provided that resolution of configuration space discretization is high enough. However, for general motion planning problems, sampled-based methods will normally sample time, the input space, and state space simultaneously. In this thesis, the resolution completeness concept is extended to all three spaces, and the first quantitative resolution complete conditions are provided for a broad class of sampling-based planners and systems. The key to resolution completeness analysis is to understand the relationship between the reachability graph, which is an intrinsic graph representation that encodes all reachable states of the given planning problem, and a search graph, which encodes states reached by a planning algorithm. Due to sampling in time, the input space, and state space, and other complications in the algorithms, mismatches are induced in the search graph. If a trajectory of a solution exists in the reachability graph, a resolution complete planner must construct a solution path that encodes the solution or it approximation in the search graph in finite time with appropriate sampling techniques and parameters.

The other two contributions are about efficient algorithm design. A symmetry-based gap reduction algorithm is designed and combined with sampling-based algorithms to solve the gap problems of these algorithms, which usually appear in their solution paths due to state space sampling, control space sampling, and algorithm design. These gaps could severely degrade the precision of the returned solutions, that is, the final states of the trajectories of the solutions might not be in the given neighborhoods of goal states. Higher precision is normally achieved using smaller gap tolerance, but this dramatically increases the computational cost. In practice, this could mean that a solution will not be found in a reasonable amount of time. In this

thesis, the performance of sampling-based algorithms is substantially improved by reducing big gaps in solution path candidates using a gap reduction technique. The gap reduction process is greatly accelerated by exploiting group symmetries of the system to avoid costly numerical integrations. Secondly, a heuristic is designed for node selection to solve the metric sensitivity problem of RRT(Rapidly-exploring Random Tree)-based planners, which means that, under differential constraints, RRT-based planners have difficulties in escaping local minima when the given metric provides a poor approximation of the true cost-to-go. Instead of designing a good metric, collision information is collected online and a value in $[0, 1]$ is assigned to each node in the search graph. The higher value a node is associated, the more is the number of trajectories, which are extended from the state of the node and have been detected in collision. Extending states associated with smaller values with higher probability would be more likely to avoid the local minima.

Chapter 2 formally defines general motion planning problems and its relationship with other specific motion planning problems, such as nonholonomic planning problem. In Chapter 3, a template for general sampling-based planning algorithms is given and several existing planning algorithms will be described in this template. Resolution completeness analysis for general sampling-based algorithms is developed in Chapter 4. The gap problems will be presented and solved in Chapter 5. In Chapter 6, the metric sensitivity problem of RRT-based algorithms will be explained and solved.

Chapter 2

Problem Formulation

A motion planning problem consists of a robotic system, work environment, initial state, and goal state. The objective is to move the system from the initial state to the goal state safely. In this thesis, all robotic systems are assumed to be mechanical systems that consist of multiple rigid bodies. The work environment is in a 3-dimensional Euclidean space and contains static obstacles and boundaries.

To solve the problem with a computer algorithm, the problem needs to be described in the computer. The problem description includes the models of the robotic system and work environment, with which an algorithm could be used to find a solution control that will move the system from the initial state to the goal state. In this chapter, the action models of the robotic systems and geometry models of the robot and work environments are first given, and then general motion planning problems are described with these models. Other types of motion planning problems, such as path planning problems and nonholonomic planning problems, are also presented as different levels of simplification of general motion planning problems.

As shown in Chapter 1, a complete characterization of a robotic system includes its geometry, sensing model, and action model. However, with the hierarchical decomposition, the action model of the robot and geometry model of the work environment and the robot will be sufficient to describe a motion planning problem.

2.1 Action Models of Robotic Systems

For motion planning problems in this thesis, the action model characterizes how the state of the system evolves with respect to the inputs. The model could be under constraints on the configuration variables and their time derivatives. To clearly describe these constraints and understand where these constraints come from, the action model is described in the order of the configuration space, kinematics, and dynamics. This order also represents different levels of simplification of the action model in the path planning problem, nonholonomic planning problem, and general motion planning problem. In the path planning problem, the action model only consider configuration space of the robot. The action model of nonholonomic planning problems includes up to kinematics. In general motion planning problems, all three parts are included in the action model.

2.1.1 Configuration space and configuration constraints

A placement of a robotic system could be characterized by a *configuration*. The set of all possible configurations is called *configuration space*, denoted as $\mathcal{C}$. For most of robotic systems, their configuration spaces can be characterized by a mathematical object, *manifold*, if a neighborhood of any configuration is homeomorphic to a Euclidean space. If the Euclidean space is of dimension n_c, then the dimension of the configuration space is also n_c and the system is said to have n_c *degrees of freedom* (DOF). A configuration is an abstract mathematical object. To use it in the computation, the configuration should be parameterized as a vector[1] in a Euclidean space. Each element of the vector is called a *configuration variable*.

One way of parameterization directly comes from the definition of the manifold, that is, each point in an n_c-dimensional configuration space can be locally represented

[1] All vectors in this thesis are assumed to be column vectors.

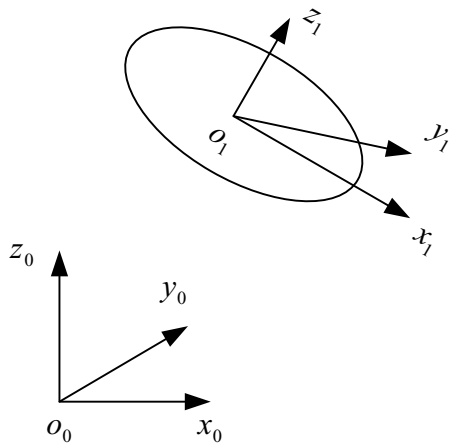


Figure 2.1: A single rigid body which moves freely in an $\mathbb{R}^3$ work environment

as a *configuration vector*, denoted as q, in $\mathbb{R}^{n_c}$. Since the number of configuration variables obtained from this method is minimum, such parameterization is called a *minimum configuration parameterization.*

For example, the configuration space of a single rigid body that could move freely in the $\mathbb{R}^3$ work environment can be characterized by a 6-dimensional manifold. To parameterize the configuration space, we can choose an inertial frame $O_0 - x_0y_0z_0$ fixed in the work environment, and a frame $O_1 - x_1y_1z_1$ fixed on the robot as shown in Fig. 2.1. A configuration could be locally represented by a configuration vector

$$q = [x, y, z, \alpha, \beta, \gamma]^T \in \mathbb{R}^6, \tag{2.1}$$

in which x, y and z are the position of the origin of the frame $O_1 - x_1y_1z_1$, and α, β and γ are Euler angles that representing the orientation of frame $O_1 - x_1y_1z_1$ with respect to frame $O_0 - x_0y_0z_0$. Euler angles normally consist of three angles, which express the orientation by sequentially rotating these three angles around different axis of a frame. For the same orientation, choosing different rotation axes could obtain different Euler angles. To be specific and concise, α, β, and γ are assumed to be yaw, pitch, and roll Euler angles which are respectively rotations around z, y and x axes.

Another parameterization is to consider an n_c-dimensional configuration space as

a submanifold[2], which is defined by a set of equality constraints in $\mathbb{R}^{n_e}$ with $n_e > n_c$. A configuration is characterized by an n_e-dimensional configuration vector

$$q = [q_1, q_2, \cdots, q_{n_e}]^T, \tag{2.2}$$

which satisfies

$$g_e^i(q) = 0 \tag{2.3}$$

for $i = 1, 2, \cdots, n_e - n_c$. These equalities g_e^i are called *holonomic constraints*, which normally correspond to the interactions between bodies in a mutli-body system, such as the linkage between bodies, or between the system and the environment, such as a robot moving on a support plane. Any n_c independent variables from the n_e configuration variables could generate a minimum configuration parameterization.

For a general robotic system that consists of k rigid bodies and moves in an $\mathbb{R}^3$ work environment, its configuration space as a submanifold in $\mathbb{R}^{6k}$ could be constructed as follows.

1. **Construct the configuration vector** Choose an inertial frame fixed in the work environment and a frame fixed on each body of the robot. The configuration of each rigid body is characterized by a configuration vector in $\mathbb{R}^6$. Therefore, a configuration vector is an element in $\mathbb{R}^{6k}$.

2. **Construct the holonomic constraints** For each interaction between the bodies of the robot or between the robot and the environment, write it as holonomic constraints on $6k$ configuration variables.

If there are n_h independent equalities constructed from holonomic constraints in the system, then the dimension of the configuration space is $6k - n_h$. This parameterization is called the *natural configuration parameterization*, which is very useful in modeling geometry of the robotic system.

[2]It is not true that every configuration space could be represented as a submanifold. For details, please refer to Chapter 4.4 in [19].

In Fig. 2.2, a robot is shown as a rectangular box that moves in a support plane. Frame $O_0 - x_0y_0z_0$ is an inertial frame fixed on the support plane, and $O_1 - x_1y_1z_1$ is a frame fixed on the robot. The configuration vector

$$q = [x, y, z, \alpha, \beta, \gamma]^T \in \mathbb{R}^6 \tag{2.4}$$

includes the position and orientation of the frame fixed on the robot with respect to the inertial frame. The constraints of moving on the support plane could be formulated in the following equations:

$$z = 0, \tag{2.5}$$

$$\beta = 0, \tag{2.6}$$

and

$$\gamma = 0, \tag{2.7}$$

in which the first equality restricts the position of the robot on the support plane, and the other two equalities ensure that the the robot will only rotate around the norm of the support plane. Therefore, the dimension of the configuration space of the robot is 3, and the independent configuration variables x, y and θ make a minimum configuration parameterization.

By adding fixed back wheels and a steerable front wheel to the robot in Fig. 2.2, a simple car system is obtained as shown in Fig. 2.3, in which $O_0 - x_0y_0z_0$ is an inertial frame fixed on the support plane, $O_1 - x_1y_1z_1$ is a frame fixed on the car, and $O_2 - x_2y_2z_2$ is a frame fixed on the front wheel. The system has two rigid bodies. From Fig. 2.2, each body has 3 holonomic constraints which are related to the supporting plane. The constraints between the front wheel and the car body are formulated as the following holonomic constraints:

$$x_2 = x_1 + l_h \cos\theta_1, \tag{2.8}$$

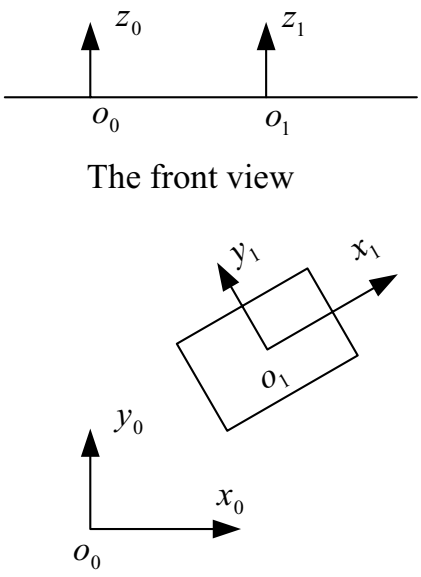


Figure 2.2: The holonomic constraints in a robot that moves in a support plane

and

$$y_2 = y_1 + l_h \sin\theta_1, \tag{2.9}$$

in which l_h is the length of O_1O_2, (x_1, y_1) and (x_2, y_2) are respectively the positions of O_1 and O_2 in frame $O_0 - x_0y_0z_0$, and θ_1 and θ_2 are respectively the yaw angle of the car body and the front wheel. With 8 holonomic constraints, the configuration space of the car system is of dimension 4. Independent configuration variables could be $(x_1, y_1, \theta_1, \theta_2)$, or $(x_2, y_2, \theta_1, \theta_2)$, each of which could be used as a minimum configuration parameterization.

Assuming that a configuration is parameterized as a configuration vector q, an inequality configuration constraint in the configuration space is in the following form:

$$g_i(q) > 0. \tag{2.10}$$

Inequality constraints usually come from the design of the system, such as joint limits, and self collision avoidance. For the car system in Fig. 2.3, the mechanical design normally requires that the absolute value of the yaw angle of the front wheel is less than a given angle, which is formulated as the following inequality configuration

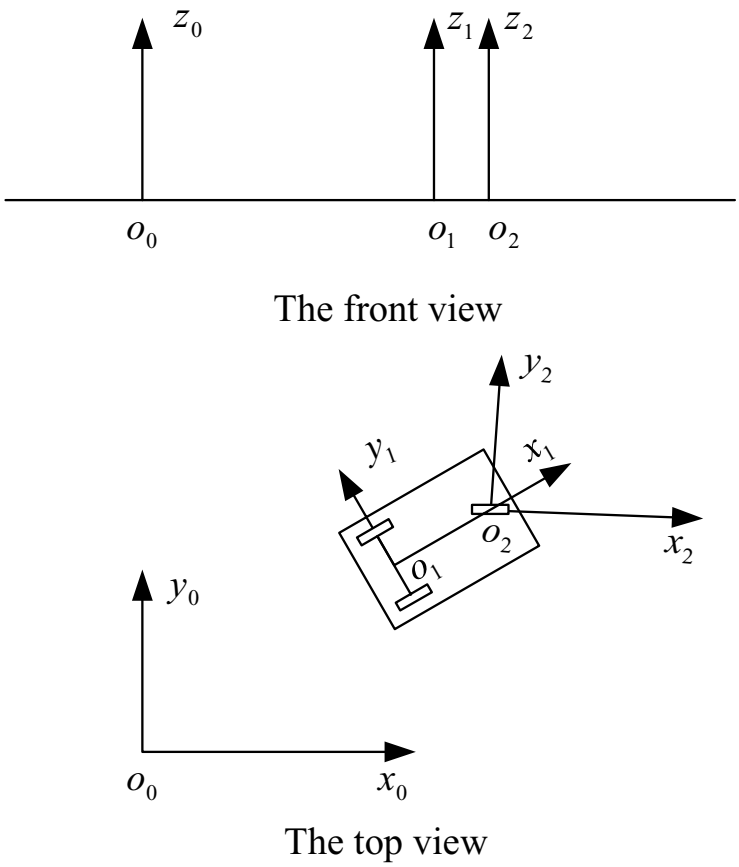


Figure 2.3: A simple car with fixed back wheels and a steerable front wheel

constraint

$$-|\theta_2| + c_\theta > 0, \tag{2.11}$$

in which c_θ is the positive constant upper bound on the steering angle.

2.1.2 Kinematics and first-order constraints

Kinematics describes the admissible velocities of the robot, which are the possible moving directions at a configuration. A configuration and a velocity at the configuration compose a *state*, which completely determines the situation of a mechanical robotic system. The set of all possible states is called the *state space*, denoted as X. The state space can be considered as a vector bundle, called the *state bundle* or *phase space*, that is a mathematical object in differential geometry. The base space of the state bundle is the configuration space of dimension n_c. The fiber space, also called

velocity space, of dimension n_v at a configuration includes all the admissible velocities at the configuration. Since the velocities satisfy the properties of vectors according to physical laws, the fiber space is an n_v-dimensional vector space. The dimension n of the state bundle is

$$n = n_c + n_v. \tag{2.12}$$

If n_v equals n_c, the state bundle is also called the *tangent bundle* of the configuration space $\mathcal{C}$, denoted as $T\mathcal{C}$, and a fiber space at a configuration $\mathbf{c}$ is called a *tangent space*, denoted as $T_{\mathbf{c}}\mathcal{C}$. Similarly to configuration, a state needs to be parameterized into a *state vector* in the Euclidean space to be used in the computation. The elements of the state vector are called *state variables*, which include configuration variables and *velocity variables*. Since the derivation of the configuration variables has been discussed in Section 2.1.1, that of the velocity variables will be discussed here. If the number of velocity variables equals the dimension of the fiber space, it is called *minimum velocity parameterization*.

With a minimum configuration parameterization of n_c-dimensional configuration space described in Section 2.1.1, a configuration $\mathbf{c}$ could be locally represented as an n_c-dimensional configuration vector

$$q = [q_1, q_2, \cdots, q_{n_c}]^T \in \mathbb{R}^{n_c}. \tag{2.13}$$

One natural way of parameterization for the velocity is to use the following set of constant unit vector fields

$$\Big\{\frac{\partial}{\partial q_1}, \frac{\partial}{\partial q_2}, \cdots, \frac{\partial}{\partial q_{n_c}}\Big\}, \tag{2.14}$$

whose values at configuration $\mathbf{c}$ represent that the corresponding configuration variables are increased at the rate of 1 unit per second. Any velocity $v_{\mathbf{c}}$ at the configuration $\mathbf{c}$ could always be decomposed into a linear combination of the values of these vector fields at configuration $\mathbf{c}$, i.e.,

$$v_{\mathbf{c}} = \dot{q}_1 \frac{\partial}{\partial q_1}(\mathbf{c}) + \dot{q}_2 \frac{\partial}{\partial q_2}(\mathbf{c}) + \cdots + \dot{q}_n \frac{\partial}{\partial q_n}(\mathbf{c}), \tag{2.15}$$

in which $\dot{q}_1$, $\dot{q}_2$, $\cdots$, and $\dot{q}_n$ are the *first-order time derivatives* of the configuration variables that represent the signed magnitude of the velocity along the direction of $\frac{\partial}{\partial q_1}(\mathbf{c})$, $\frac{\partial}{\partial q_2}(\mathbf{c})$, $\cdots$, and $\frac{\partial}{\partial q_n}(\mathbf{c})$. Therefore, the velocity $v_{\mathbf{c}}$ can be represented by a *velocity vector*

$$\dot{q} = [\dot{q}_1, \dot{q}_2, \cdots, \dot{q}_{n_c}]^T \in \mathbb{R}^{n_c}, \tag{2.16}$$

which is called a *natural velocity parameterization*.

A *first-order nonholonomic constraint* could be formulated as the following equality of configuration variables and their first-order time derivatives:

$$h_e(q, \dot{q}) = 0, \tag{2.17}$$

if h_e cannot be transformed into

$$g_e(q) = 0, \tag{2.18}$$

i.e., a holonomic constraint.

If the back wheels cannot slip on the ground, the system in Fig. 2.4 has a first-order nonholonomic constraint. In Fig. 2.3, a configuration of the car could be parameterized by configuration variables x_1, y_1, θ_1 and θ_2. The value of the slipping velocity of the back wheels can be formulated as the following equation of the first-order time derivatives of the configuration variables

$$v_y = \dot{x}_1 \sin\theta_1 - \dot{y}_1 \cos\theta_1. \tag{2.19}$$

Therefore, the nonslipping constraint of the back wheels encodes the following first-order nonholonomic constraint

$$\dot{x}_1 \sin\theta_1 - \dot{y}_1 \cos\theta_1 = 0. \tag{2.20}$$

If there are no first-order nonholonomic constraints in a robotic system, a natural velocity parameterization is also a minimum velocity parameterization. As shown in Fig. 2.1, since the system can move freely in $\mathbb{R}^3$, a configuration could be locally

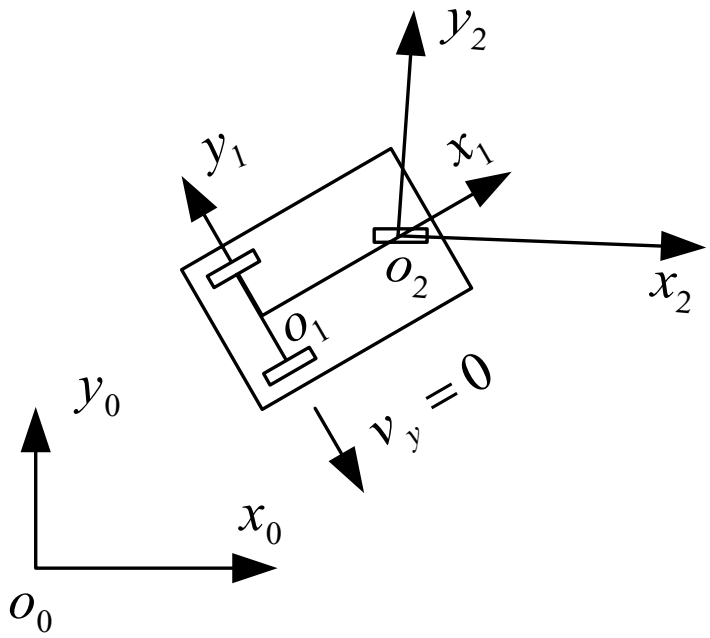


Figure 2.4: A system with a first-order nonholonomic constraint due to the nonslipping back wheels

represented as a configuration vector

$$q = [x, y, z, \alpha, \beta, \gamma]^T \in \mathbb{R}^6 \tag{2.21}$$

using an inertial frame fixed in the work environment and a frame fixed on the robot. With the same frames, the robot could translate along or rotate around each axis to independently change each configuration variables. Therefore, the space of admissible velocities at the configuration is 6-dimensional vector space. Using the following constant vector fields

$$\left\{\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}, \frac{\partial}{\partial \alpha}, \frac{\partial}{\partial \beta}, \frac{\partial}{\partial \gamma}\right\}, \tag{2.22}$$

a velocity could be represented as a velocity vector

$$[\dot{x}, \dot{y}, \dot{z}, \dot{\alpha}, \dot{\beta}, \dot{\gamma}]^T \in \mathbb{R}^6, \tag{2.23}$$

whose elements are the first-order time derivative of configuration variables.

When first-order nonholonomic constraints exist, one way of parameterization is to implicitly represent a velocity at a configuration by the first-order time derivatives

of n_c configuration variables and these nonholonomic constraints. Another way that leads to a minimum velocity parameterization is to find a set of n_v vector fields

$$\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_{n_v}\}, \tag{2.24}$$

in which the natural velocity parameterization of velocity $\mathbf{x}_i(\mathbf{c})$ satisfies the nonholonomic constraints in Eq. (2.17) at the configuration $\mathbf{c}$ for $i = 1, 2, \cdots, n_v$. Theoretically, since the state bundle with n_v-dimensional fiber space is also a manifold, a set of n_v linearly independent vector fields always exists. Therefore, a velocity $v_{\mathbf{c}}$ at a configuration $\mathbf{c}$ has the following linear decomposition

$$v_{\mathbf{c}} = v_1\mathbf{x}_1(c) + v_2\mathbf{x}_2(c) + \cdots + v_{n_v}\mathbf{x}_k(c), \tag{2.25}$$

and can be represented as a velocity vector

$$v = [v_1, v_2, \cdots, v_{n_v}]^T \in \mathbb{R}^{n_v}. \tag{2.26}$$

Such parameterization is called a *general velocity parameterization*.

For the system in Fig.2.4, the following three vector fields satisfy the nonholonomic constraint in Eq. (2.20).

$$\mathbf{x}_1(x_1, y_1, \theta_1, \theta_2) = \frac{\partial}{\partial\theta_2}, \tag{2.27}$$

$$\mathbf{x}_2(x_1, y_1, \theta_1, \theta_2) = \frac{\partial}{\partial\theta_1}, \tag{2.28}$$

and

$$\mathbf{x}_3(x_1, y_1, \theta_1, \theta_2) = \cos\theta_1\frac{\partial}{\partial x_1} + \sin\theta_1\frac{\partial}{\partial y_1}, \tag{2.29}$$

in which $\mathbf{x}_1$ and $\mathbf{x}_2$ respectively represent pure rotation to change the yaw angles of the car body and the steering wheel with the unit speed, and $\mathbf{x}_3$ represents the pure translation along x_1 axis of $O_1 - x_1y_1z_1$ with the unit speed. It can be easily verified that these vector fields satisfy the nonholonomic constraints in Eq. (2.20). Since the velocity space is 3-dimensional and the values of these vector fields are

linear independent at any configuration, any velocity $v_{\mathbf{c}}$ at configuration $\mathbf{c}$ could be represented as the following linear composition

$$v_{\mathbf{c}} = v_1 \mathbf{x}_1(c) + v_2 \mathbf{x}_2(c) + v_3 \mathbf{x}_3(c), \tag{2.30}$$

in which v_1, v_2, and v_3 are velocity variables.

A *first-order inequality constraint* could be formulated as the following inequality of configuration vector q and velocity vector v

$$h_i(q, v) > 0. \tag{2.31}$$

For example, if the velocity of the car along x_1 axis of $O_1 - x_1 y_1 z_1$ in Fig. 2.4 is required to be larger than 0 and less than 5, then it can be formulated as the following first-order inequality constraints:

$$v_3 > 0, \tag{2.32}$$

and

$$5 - v_3 > 0. \tag{2.33}$$

2.1.3 Dynamics and second-order constraints

Dynamics of a system shows the relationship between first-order time derivatives of the velocity variables and the inputs applied on the system, which could be derived with the Newton-Euler mechanics, Lagrangian mechanics, or Hamiltonian mechanics. In this thesis, the dynamics for general systems will be provided without the details of the derivation. Please refer to [51] for details. Assume that the state bundle of a general system has n_c-dimensional configuration space $\mathcal{C}$ and n_v-dimensional velocity space, and a configuration $\mathbf{c}$ is represented as a configuration vector

$$q = [q_1, q_2, \cdots, q_{n_c}]^T \in \mathbb{R}^{n_c} \tag{2.34}$$

by a minimum configuration parameterization. The description of its dynamics also needs the following objects:

- Kinetic energy as a Riemannian metric $\mathcal{M}$ defined on configuration space $\mathcal{C}$. At any configuration $\mathbf{c}$ in the configuration space $\mathcal{C}$, $\mathcal{M}$ is an inner product, denoted as $\mathcal{M}_{\mathbf{c}}$, which is defined on the tangent space $T_{\mathbf{c}}\mathcal{C}$

$$\mathcal{M}_{\mathbf{c}} : T_{\mathbf{c}}\mathcal{C} \times T_{\mathbf{c}}\mathcal{C} \to \mathbb{R}^+ \cup \{0\}. \tag{2.35}$$

If a system has velocity $v_{\mathbf{c}}$ at configuration $\mathbf{c}$, then the kinetic energy of the system will be $\mathcal{M}_{\mathbf{c}}(v_{\mathbf{c}}, v_{\mathbf{c}})$.

Assume that the velocity $v_{\mathbf{c}}$ in the tangent space is represented under the natural velocity parameterization as a vector

$$\dot{q} = [\dot{q}_1, \dot{q}_2, \cdots, \dot{q}_{n_c}]^T \in \mathbb{R}^{n_c}. \tag{2.36}$$

Since $\mathcal{M}_{\mathbf{c}}$ is a parameterized inner product, it could be represented as an $n_c \times n_c$ matrix, denoted as $[m_{ij}(q)]_{n_c \times n_c}$, during the computation. The kinetic energy will be calculated as

$$\begin{aligned} \mathcal{M}_{\mathbf{c}}(v_{\mathbf{c}}, v_{\mathbf{c}}) &= [\dot{q}_1, \dot{q}_2, \cdots, \dot{q}_{n_c}][m_{ij}(q)]_{n_c \times n_c}[\dot{q}_1, \dot{q}_2, \cdots, \dot{q}_{n_c}]^T \\ &= \sum_{i=1}^{n_c}\sum_{j=1}^{n_c} m_{ij}(q)\dot{q}_i\dot{q}_j. \end{aligned} \tag{2.37}$$

The inverse matrix of $[m_{ij}]_{n_c \times n_c}$ is represented as $[m^{ij}]_{n_c \times n_c}$.

For the car in Fig. 2.4, assume that the mass and inertial of the car body are respectively M and I_1, the steering wheel has inertial I_2, the mass of the steering wheel is ignored, and a velocity $v_{\mathbf{c}}$ at configuration $\mathbf{c}$ in the natural parameterization is a vector

$$\dot{q} = [\dot{x}_1, \dot{y}_1, \dot{\theta}_1, \dot{\theta}_2]^T \in \mathbb{R}^4, \tag{2.38}$$

and $\mathcal{M}_{\mathbf{c}}$ is of the following form:

$$\mathcal{M}_{\mathbf{c}} = \begin{bmatrix} \frac{m}{2} & 0 & 0 & 0 \\ 0 & \frac{m}{2} & 0 & 0 \\ 0 & 0 & \frac{I_1+I_2}{2} & \frac{I_2}{2} \\ 0 & 0 & \frac{I_2}{2} & \frac{I_2}{2} \end{bmatrix}. \tag{2.39}$$

The kinetic energy will be calculated as

$$\begin{aligned}\mathcal{M}_{\mathbf{c}}(v_{\mathbf{c}}, v_{\mathbf{c}}) &= [\dot{x}_1, \dot{y}_1, \dot{\theta}_1, \dot{\theta}_2] \begin{bmatrix} \frac{m}{2} & 0 & 0 & 0 \\ 0 & \frac{m}{2} & 0 & 0 \\ 0 & 0 & \frac{I_1+I_2}{2} & \frac{I_2}{2} \\ 0 & 0 & \frac{I_2}{2} & \frac{I_2}{2} \end{bmatrix} \begin{bmatrix} \dot{x}_1 \\ \dot{y}_1 \\ \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} \\ &= \frac{m}{2}\dot{x}_1^2 + \frac{m}{2}\dot{y}_1^2 + \frac{I_1+I_2}{2}\dot{\theta}_1^2 + \frac{I_2}{2}\dot{\theta}_2^2 + I_2\dot{\theta}_1\dot{\theta}_2. \end{aligned} \tag{2.40}$$

- Potential energy as a function V from the configuration space to $\mathbb{R}$.

  By choosing a reference plane for which the potential energy is zero, the potential energy of a system with mass M at configuration $\mathbf{c}$ could be calculated as

  $$V(q) = Mgh(q), \tag{2.41}$$

  in which g is the gravitational acceleration of the earth, and $h(q)$ is a function of configuration whose value is the signed distance of the mass center of the system above the reference plane.

- A set of mutually orthonormal vector fields

  $$\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_{n_v}\}, \tag{2.42}$$

  in which

  $$\mathcal{M}_{\mathbf{c}}(\mathbf{x}_i(\mathbf{c}), \mathbf{x}_j(\mathbf{c})) = 0 \tag{2.43}$$

  for $i \neq j$ and $i, j = 1, 2, \cdots, n_v$ and any $\mathbf{c}$ in $\mathcal{C}$. Using the set, the general velocity parameterization of a velocity $v_{\mathbf{c}}$ at configuration $\mathbf{c}$ is a vector

  $$v = [v_1, v_2, \cdots, v_{n_v}]^T \in \mathbb{R}^{n_v}. \tag{2.44}$$

  Note that such mutually orthonormal vector fields could be derived from any set of n_v linearly independent vector fields using Gram-Schmidt orthonormal-

ization. Also note that dynamics could be described with any set of n_v linearly independent vector fields, but the formulation could be complicated.

- General forces as one-forms.

  The inputs for an actual robotic system are normally forces. A force could generate both translational forces and rotational torques, which together are called a *general force* of the force.

  Since general forces satisfy the properties of vectors and the multiplication of a general force and velocity could generate a scalar that is the power by physical laws, the general force at a configuration could be considered as a covector that maps a velocity vector into a scalar. Therefore, general forces could be represented as one-forms. Choosing the following set of n_c constant one-forms

$$\{dq_1, dq_2, \cdots, dq_{n_c}\}, \tag{2.45}$$

  in which $\frac{d}{dq_i}$ will generate a unit power when applying along the corresponding constant vector field $\frac{\partial}{\partial q_i}$ for $i = 1, 2, \cdots, n_c$, a general force $\mathcal{F}$ at configuration $\mathbf{c}$ can be decomposed into the following linear combination

$$\mathcal{F}(\mathbf{c}) = q^1(\mathbf{c})\frac{d}{dq_1}(\mathbf{c}) + q^2(\mathbf{c})\frac{d}{dq_2}(\mathbf{c}) + \cdots + q^{n_c}(\mathbf{c})\frac{d}{dq_{n_c}}(\mathbf{c}), \tag{2.46}$$

  and represented in the following vector form

$$\mathbf{F}(q) = [q^1(q), q^2(q), \cdots, q^{n_c}(q)]^T \in \mathbb{R}^{n_c}. \tag{2.47}$$

  Assume that the system has m forces, the general force of each unit forces are represented as

$$\mathbf{F}_1, \mathbf{F}_2, \cdots, \mathbf{F}_m, \tag{2.48}$$

  and the value of forces are represented as an *input vector*

$$u = [u_1, u_2, \cdots, u_m]^T \in \mathbb{R}^m. \tag{2.49}$$

If the number of forces is less than the dimension of the configuration space, the system is called an *underactuated* system.

The general dynamics equation for the above system is as follows

$$\dot{v}_l + \sum_{i=1}^{n_v}\sum_{j=1}^{n_v}\tilde{\Gamma}_{ij}^l v_i v_j = \sum_{a=1}^{m} Y_a^l u_a - V^l, l = 1, 2, \cdots, n_v, \tag{2.50}$$

in which

$$\tilde{\Gamma}_{ij}^l = \frac{\mathcal{M}(\nabla_{\mathbf{x}_i}\mathbf{x}_j, \mathbf{x}_k)}{\mathcal{M}(\mathbf{x}_k, \mathbf{x}_k)}, \tag{2.51}$$

$$\nabla_{\mathbf{x}_i}\mathbf{x}_j = \sum_{l=1}^{n_v}\Gamma_{ij}^l \mathbf{x}_l \tag{2.52}$$

is the covariant derivative of $\mathbf{x}_j$ with respect to $\mathbf{x}_i$,

$$\Gamma_{ij}^l = \frac{1}{2}\sum_{p=1}^{n_v} m^{ql}\left(\frac{\partial m_{ip}}{\partial q_j} + \frac{\partial m_{jp}}{\partial q_i} - \frac{\partial m_{ij}}{\partial q_p}\right), \tag{2.53}$$

$$Y_a^l = \frac{\mathcal{M}(Y_a, \mathbf{x}_l)}{\mathcal{M}(\mathbf{x}_l, \mathbf{x}_l)}, \tag{2.54}$$

$$Y_a = ([m^{ij}]_{n_c \times n_c}\mathbf{F}_a)^T\left[\frac{\partial}{\partial q_1}, \frac{\partial}{\partial q_2}, \cdots, \frac{\partial}{\partial q_{n_c}}\right]^T \tag{2.55}$$

is the vector field obtained from the general force $\mathbf{F}_a$,

$$V^l = \frac{\mathcal{M}(\nabla V, \mathbf{x}_l)}{\mathcal{M}(\mathbf{x}_l, \mathbf{x}_l)}, \tag{2.56}$$

and

$$\nabla V = \frac{\partial V}{\partial q_1}\frac{\partial}{\partial q_1} + \frac{\partial V}{\partial q_2}\frac{\partial}{\partial q_2} + \cdots + \frac{\partial V}{\partial q_{n_c}}\frac{\partial}{\partial q_{n_c}} \tag{2.57}$$

is the vector field obtained by applying the gradient operator ∇ on V.

A *second-order nonholonomic constraint* is in the form the following equality

$$k_e(q, v, \dot{v}) = 0, \tag{2.58}$$

in which

$$\dot{v} = [\dot{v}_1, \dot{v}_1, \cdots, \dot{v}_{n_v}]^T \in \mathbb{R}^{n_v}, \tag{2.59}$$

if k_e cannot be transformed into

$$h_e(q, v) = 0 \tag{2.60}$$

and

$$g_e(q) = 0. \tag{2.61}$$

For example, if the right side of Eq. (2.50) is 0 for some $l = 1, 2, \cdots$, or n_v, then it could induce a second-order nonholonomic constraint in the following form

$$\dot{v}_l + \sum_{i=1}^{n_v} \sum_{j=1}^{n_v} \tilde{\Gamma}_{ij}^{l} v_i v_j = 0, \tag{2.62}$$

which often appears in a manipulator robot with passive joints.

A *second-order inequality constraint* is in the form of the following equality

$$k_i(q, v, \dot{v}) > 0, \tag{2.63}$$

which normally comes from safety considerations for the system. For example, if the forces generate a very large acceleration, then the force might not be enough to hold different parts of the robotic system together such that the system will be disassembled. By substituting $\dot{v}$ with terms of u, q, and v according to Eq. (2.50), these inequality constraints could be transformed into the following equalities:

$$k_i(q, v, u) > 0. \tag{2.64}$$

Note that if $\dot{v}$ in a second-order inequality constraint of Eq. (2.63) is only functions of q and v, such as in the case of the second-order nonholonomic constraints, then the inequality constraint is actually a first-order inequality constraint by substituting the accelerations with the terms of q and v.

2.2 Geometry Models of the Robot and Work Environment

The geometry model of a system serves the purpose of determining the space occupancy of the system, with which the algorithm could know whether the system at a given configuration collides with itself or objects in the work environment. A geometry model of the system includes a closed set

$$G_r \subset W \tag{2.65}$$

and rigid transformation

$$T_G : \mathcal{C} \times 2^W \to 2^W, \tag{2.66}$$

in which

$$W = \mathbb{R}^3. \tag{2.67}$$

The set G_r is associated with a fixed configuration, at which any point on the robot is in G_r, the transformation T_G will translate and rotate G_r according to a given configuration into a new set $G_r{}'$ such that any point on the robot at the given configuration is in $G_r{}'$.

Normally, the set G_r is described with a set of geometric primitives, such as lines, curves, planes, and surfaces. For example, with a set of plane primitives, a convex set could be constructed as the intersection of the half-spaces, each of which corresponds to a surface primitive, and a nonconvex set could be constructed as the union of multiple convex sets formed with surface primitives. In some cases, if the set G_r is very simple, it could be characterized by simple inequalities. For example, if the robot has a shape of a ball of radius 1 in $\mathbb{R}^3$, then G_r could be characterized as

$$x^2 + y^2 + z^2 \leq 1, \tag{2.68}$$

with its associated position at $(0, 0, 0)$. For a system that consists of l rigid bodies,

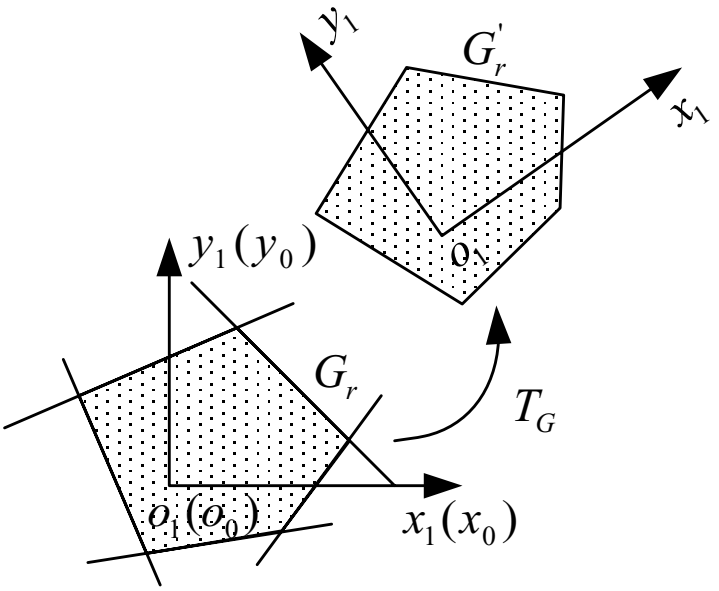


Figure 2.5: The geometry model of a robot

the transformation T_G could be defined by l matrices in $SE(3)$, each of which will transform the shape of a rigid body according to the current configuration.

An example of the geometry of a robotic system is given in Fig. 2.5, in which G_r at configuration $(0, 0, 0)$ is presented by the intersection of 5 half-planes each of which is represented by a straight line, and the transformation T_G for configuration (x, y, θ) is represented by a matrix

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & x \\ \sin\theta & \cos\theta & 0 & y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{2.69}$$

which transforms G_r to $G_r{}'$.

The work environment is represented by only a closed set

$$G_w \subset \mathbb{R}^3 \tag{2.70}$$

since the environment is assumed to be static in the thesis. The similar techniques to represent G_r for geometry of the robot could be also used.

2.3 General Motion Planning Problems

These problems are also called kinodynamic planning problems. For a given general motion planning problem, denoted as $\mathcal{P}$, its work environment is represented by a geometry model G_w. Assume that a system has an n_c-dimensional configuration space, l_c configuration constraints, l_n first-order nonholonomic constraints, l_v first-order inequality constraints, l_a second-order inequality constraints, m forces with

$$m \leq n_c, \tag{2.71}$$

and geometry model G_r and T_G. The dimension n_v of the velocity space is

$$n_v = n_c - l_n, \tag{2.72}$$

and the dimension n of the state bundle is

$$n = n_v + n_c. \tag{2.73}$$

Assume that a minimum configuration and general velocity parameterizations are used, a state is locally parameterized as a state vector (called state for short later)

$$x = [q^T, v^T] \in \mathbb{R}^n, \tag{2.74}$$

in which

$$q \in \mathbb{R}^{n_c} \tag{2.75}$$

is the configuration vector, called configuration for short later, and

$$v \in \mathbb{R}^{n_v} \tag{2.76}$$

is the velocity vector, called velocity for short later, by using a set of vector fields

$$\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_{n_v}\}. \tag{2.77}$$

The general motion planning problem, $\mathcal{P}$, is represented as a tuple,

$$(X, X_{obs}, U, \bar{\mathcal{U}}, f, x_{init}, X_g), \tag{2.78}$$

in which X and X_{obs} are the state space and its obstacles, U is the input space, $\bar{\mathcal{U}}$ is the control space generator set, f is the motion equation encoding both kinematics and dynamics, x_{init} is the initial state, and X_g is the goal state set. These symbols will be further explained in details in the following sections.

2.3.1 State space and its obstacles

The state space, X, is the state bundle of the robotic system. In this thesis, X is required to be a bounded and open differentiable manifold of dimension n. The configuration space is n_c-dimensional and the velocity space is n_v-dimensional. Assume $\|\cdot\|$ is a given norm defined on X.

The closed set

$$X_{obs} \subset X \tag{2.79}$$

encodes the collision avoidance constraints from the work environment, configuration inequality constraints, and first-order inequality constraints, and is defined as follows:

$$X_{obs} = X_{col} \cup X_c \cup X_v, \tag{2.80}$$

in which the set

$$X_{col} = \{x \in X \mid T_G(q, G_r) \cap G_w \neq \emptyset\} \tag{2.81}$$

includes all states that cause the collision between the robot and work environment, the set

$$X_c = \bigcup_{l=1}^{l_c} \{x \in X \mid g_i^l(q) \not> 0\} \tag{2.82}$$

includes all states that violate the configuration inequality constraints from the system, and the set

$$X_v = \bigcup_{l=1}^{l_v} \{x \in X \mid h_i^l(q, v) \not> 0\} \tag{2.83}$$

includes all states that violate the first-order inequality constraints from the system. The set X_{col} from the environment is often called the *global constraints*, and X_c and

X_v from the system are called the *local constraints*. For convenience, an open set X_{free} is defined as

$$X_{free} = X \setminus X_{obs}, \tag{2.84}$$

which is called the *violation-free* set.

2.3.2 The input space and control space

The *input space* U includes all possible input vectors, called *input* for short later, and is defined as a rectangular subset of $\mathbb{R}^m$:

$$U = [u_1^{\min}, u_1^{\max}] \times [u_2^{\min}, u_2^{\max}] \times \cdots \times [u_m^{\min}, u_m^{\max}], \tag{2.85}$$

in which $u_i^{\min}$ and $u_i^{\max}$ are the lower and upper bounds on the value of the i-th input force, and each input u in U also satisfies l_a second-order inequality constraints

$$k_i^l(x, u) = k_i^l(q, v, u) > 0 \tag{2.86}$$

for $l = 1, 2, \cdots$, and l_a. It is assumed that U is equipped with a norm $\|\cdot\|$.

The *control space*, $\mathcal{U}$, of $\mathcal{P}$ contains all possible controls for $\mathcal{P}$. It is important to keep in mind that an input represents a value in $\mathbb{R}^m$ and a *control*[3], denoted as $\tilde{u}$, represents a piecewise-continuous vector-valued function from $[0, t]$ to U for some real positive t. An example of a control

$$\tilde{u} : [0, t_f] \rightarrow U \tag{2.87}$$

is shown in Fig. 2.6, in which the value of the control $\tilde{u}$ at any time t_1 in $[0, t_f]$ is in the input space U. For any $\tilde{u}_1$ and $\tilde{u}_2$ in $\mathcal{U}$, their *distance* is defined as

$$\rho(\tilde{u}_1, \tilde{u}_2) = \sup_{s \in [0, \min\{\bar{t}(\tilde{u}_1), \bar{t}(\tilde{u}_2)\}]} \|\tilde{u}_1(s) - \tilde{u}_2(s)\|. \tag{2.88}$$

[3]In [52], the term control is also used to represent an input function.

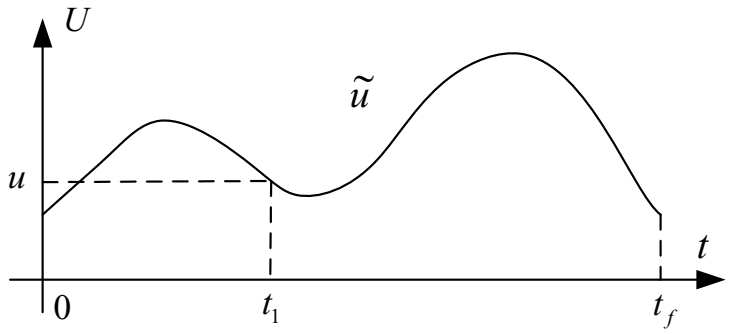


Figure 2.6: An example of the control $\tilde{u}$ as a function from a time interval to the input space

To define the sample space from which the controls are sampled for sampling-based planning algorithms, $\mathcal{U}$ is defined in this thesis as a *semigroup*[4] generated from a nonempty *control space generator set*, $\bar{\mathcal{U}}$, which is a set of continuous controls and will be used to construct the sample space for sampling-based planning algorithms. The controls in the set could be arbitrary functions or just a special class of functions, such as constant functions or sinusoidal functions. The number of controls in the set could be either countable or uncountable. The set could be either continuous or discrete. For general robotic systems, the generator set is uncountable and continuous, and consists of arbitrary functions. For convenience, let the *control duration range*, $\mathcal{D}$, be defined as[5]

$$\mathcal{D} = [\inf_{\tilde{u} \in \bar{\mathcal{U}}} \bar{t}(\tilde{u}), \sup_{\tilde{u} \in \bar{\mathcal{U}}} \bar{t}(\tilde{u})] \subset [0, \infty), \tag{2.89}$$

in which

$$\bar{t} : \mathcal{U} \to (0, \infty) \tag{2.90}$$

gives the time duration of any $\tilde{u}$ in $\mathcal{U}$. The semigroup operation $\circ$ for any $\tilde{u}_1$ and $\tilde{u}_2$

[4] A semigroup is a set equipped with an operation whose elements satisfy closure and associativity properties, but need not have inverses.

[5] Even though the infimum could be 0, the duration of any control in $\bar{\mathcal{U}}$ is required to be bigger than 0.

in $\mathcal{U}$ is defined as

$$(\tilde{u}_1 \circ \tilde{u}_2)(t) = \begin{cases} \tilde{u}_1(t) & t \in [0, \bar{t}(\tilde{u}_1)) \\ \\ \tilde{u}_2(t - t_1) & t \in [\bar{t}(\tilde{u}_1), \bar{t}(\tilde{u}_1) + \bar{t}(\tilde{u}_2)]. \end{cases} \tag{2.91}$$

The *k-order expanded control set* is defined as

$$\hat{\mathcal{U}}^k = \{\tilde{u}_1 \circ \tilde{u}_2 \circ \cdots \circ \tilde{u}_k \mid \tilde{u}_i \in \bar{\mathcal{U}} \text{ for } i = 1, 2, \cdots, k\}. \tag{2.92}$$

The *k-order generated control set* is defined as

$$\bar{\mathcal{U}}^k = \bigcup_{i=1}^{k} \hat{\mathcal{U}}^i. \tag{2.93}$$

The control space is now defined as

$$\mathcal{U} = \bar{\mathcal{U}}^\infty. \tag{2.94}$$

It is clear that

$$\bar{\mathcal{U}}^{k-1} \subset \bar{\mathcal{U}}^k. \tag{2.95}$$

2.3.3 Motion equation to encode kinematics and dynamics

To facilitate computation of motion of the robotic system in the computer, the kinematics and dynamics of the system is normally represented as a set of input-parameterized Ordinary Differential Equations (ODEs), called *motion equation* and denoted as f

$$\dot{x} = f(x, u), \tag{2.96}$$

in which x is the state, $\dot{x}$ is the first-order time derivative of x, and u in the input of the system. In this thesis, the motion equation is independent of time, i.e., the system is *time-invariant*. Time-invariant systems are also known as *autonomous* systems.

A *general form* of a *trajectory* is

$$\tau : [0, t] \to X, \tag{2.97}$$

i.e., a function from a time interval to the state space, which could be either continuous or discontinuous in time. Given a control

$$\tilde{u} : [0, \bar{t}(\tilde{u})] \to U \tag{2.98}$$

and a starting state x_i, a *trajectory of the control* $\tilde{u}$ from state x_i could be also defined as a state transition map

$$\tau_{\tilde{u}} : X \times [0, \bar{t}(\tilde{u})] \to X, \tag{2.99}$$

which is obtained by the following integration

$$\tau_{\tilde{u}}(x_i, t) = x_i + \int_0^t f(x(\tau), \tilde{u}(\tau))\, d\tau. \tag{2.100}$$

It is easy to see that

$$\tau_{\tilde{u}}(x_i, 0) = x_i. \tag{2.101}$$

Given state x_i and control $\tilde{u}$, the trajectory of control $\tilde{u}$ from x_i in the general form is

$$\tau_{\tilde{u}}(x_i, \cdot) : [0, \bar{t}(\tilde{u})] \to X. \tag{2.102}$$

For a trajectory $\tau_{\tilde{u}}(x_i, \cdot)$ of control $\tilde{u}$ from starting state x_i, if for any t in $[0, \bar{t}(\tilde{u})]$

$$\tau_{\tilde{u}}(x_i, t) \in X_{free} \tag{2.103}$$

and

$$k_i^l(\tau_{\tilde{u}}(x_i, t), \tilde{u}(t)) > 0 \tag{2.104}$$

for $l = 1, 2, \cdots, l_a$, the trajectory is called *violation-free*. Let the *w-tube of a trajectory*

$$\tau : [0, t_f] \to X, \tag{2.105}$$

be defined as

$$T_\tau^w = \{x \mid \|x - \tau(t)\| \leq w,\ \forall t \in [0, t_f]\}, \tag{2.106}$$

in which $\|\cdot\|$ is a norm on the state space. A violation-free trajectory τ is called to have a *clearance* w if for any $\tilde{u}$ and state x_i

$$\tau_{\tilde{u}}(x_i, t) \in T_\tau^w, t \in [0, \bar{t}(\tilde{u})] \tag{2.107}$$

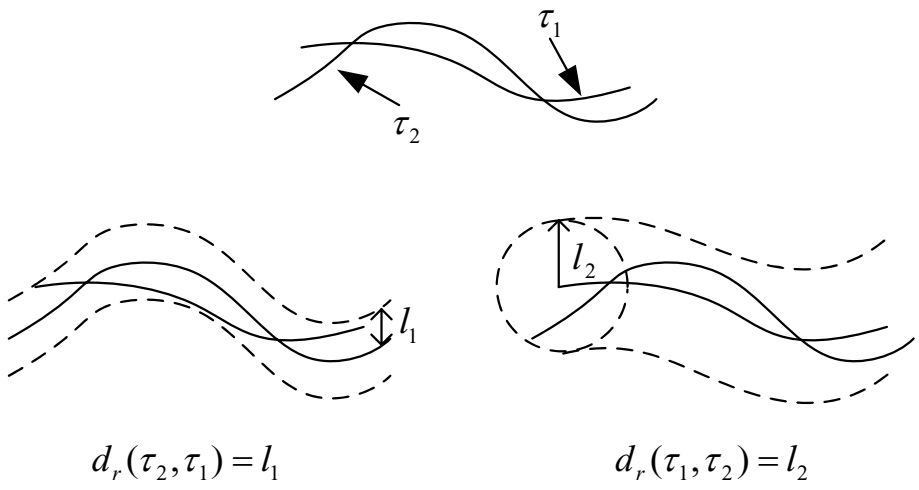


Figure 2.7: The distance from trajectory τ_2 to τ_1 does not equal that from τ_1 to τ_2.

implies that the trajectory of $\tilde{u}$ from x_i is violation-free, i.e., if the image of a trajectory of a control from a state is in w-tube of trajectory τ with clearance w, then the trajectory is violation-free. If there are no second-order inequality constraints, then the conditions for trajectories to be violation-free or have clearance w will be just respectively requiring the image or w-tube of the trajectory in X_{free}.

Given two trajectories

$$\tau_2 : [0, t_2] \to X \tag{2.108}$$

and

$$\tau_1 : [0, t_1] \to X \tag{2.109}$$

which are respectively trajectories of $\tilde{u}_1$ and $\tilde{u}_2$ from x_1 and x_2, the distance $d_\tau(\tau_2, \tau_1)$ from trajectory τ_2 to τ_1 is defined as

$$d_\tau(\tau_2, \tau_1) = \sup_{s_2 \in [0,t_2]} \inf_{s_1 \in [0,t_1]} \|\tau_1(s_1) - \tau_2(s_2)\|. \tag{2.110}$$

If $d_\tau(\tau_2, \tau_1)$ is l_1, then the image of trajectory τ_1 will be in the l_1-tube of trajectory τ_2. The distance from trajectory τ_2 to τ_1 could be different from that from trajectory τ_1 to τ_2 as shown in Fig. 2.7. Control $\tilde{u}_2$ (or trajectory τ_2) is an *η-neighboring control (or trajectory)*[6] of $\tilde{u}_1$ (or τ_1) if the image of τ_2 is in η-tube of τ_1. The concatenation

[6]Note that the definition requires only that τ_2 stays within a "tube" that surrounds τ_1, without

operator $\circ$ for two trajectories is defined as

$$(\tau_1 \circ \tau_2)(t) = \begin{cases} \tau_1(t) & t \in [0, t_1) \\ \\ \tau_2(t - t_1) & t \in [t_1, t_1 + t_2]. \end{cases} \tag{2.111}$$

Assume that $\tau_{\tilde{u}_1}(x_1, \cdot)$ is the trajectory of control $\tilde{u}_1$ from state x_1, $\tau_{\tilde{u}_2}(x_2, \cdot)$ is the trajectory of control $\tilde{u}_2$ from state x_2, and $\tau_{\tilde{u}}(x_1, \cdot)$ is the trajectory of control $\tilde{u}$ from state x_1 in which

$$\tilde{u} = \tilde{u}_1 \circ \tilde{u}_2. \tag{2.112}$$

Only when

$$x_2 = \tau_{\tilde{u}_1}(x_1, \bar{t}(\tilde{u}_1)), \tag{2.113}$$

i.e., the trajectory $\tau_{\tilde{u}_2}$ of control $\tilde{u}_2$ is from the final state of trajectory $\tau_{\tilde{u}_1}(x_1, \cdot)$, is the following equation derived

$$\tau_{\tilde{u}}(x_1, t) = \tau(t), \tag{2.114}$$

in which

$$\tau = \tau_{\tilde{u}_1}(x_1, \cdot) \circ \tau_{\tilde{u}_2}(x_2, \cdot), \tag{2.115}$$

and

$$t \in [0, \bar{t}(\tilde{u}_1) + \bar{t}(\tilde{u}_2)]. \tag{2.116}$$

From the description in Sections 2.1.2 and 2.1.3, the general motion equation of a robotic system is defined as

$$\dot{x} = \begin{bmatrix} \dot{q} \\ \\ \dot{v} \end{bmatrix} = \begin{bmatrix} f_q(q, v) \\ \\ f_v(q, v, u) \end{bmatrix}, \tag{2.117}$$

in which

$$f_q(q, v) = \sum_{k=1}^{n_v} v_k \mathbf{x}_k(q), \tag{2.118}$$

any concern for the times at which the trajectories reach various states. This is allowed because this analysis is primarily concerned with the feasibility of trajectories, as opposed to optimality.

and

$$f_v = \begin{bmatrix} -\sum_{i=1}^{n_v}\sum_{j=1}^{n_v}\tilde{\Gamma}_{ij}^1 v_i v_j + \sum_{a=1}^{m} Y_a^1 u_a - V^1 \\ -\sum_{i=1}^{n_v}\sum_{j=1}^{n_v}\tilde{\Gamma}_{ij}^2 v_i v_j + \sum_{a=1}^{m} Y_a^2 u_a - V^2 \\ \cdots \\ -\sum_{i=1}^{n_v}\sum_{j=1}^{n_v}\tilde{\Gamma}_{ij}^{n_v} v_i v_j + \sum_{a=1}^{m} Y_a^{n_v} u_a - V^{n_v} \end{bmatrix}, \tag{2.119}$$

in which terms are defined in Eq. (2.50). These equations show the derivation of motion equations from kinematics and dynamics of robotic systems.

2.3.4 Solutions

The *initial state* and *goal state set* are respectively

$$x_{init} \in X_{free} \tag{2.120}$$

and

$$X_g \subset X_{free}. \tag{2.121}$$

A control $\tilde{u}$ is an *exact solution* to $\mathcal{P}$ if its trajectory $\tau_{\tilde{u}}(x_{init}, \cdot)$ from the initial state x_{init} is violation-free and

$$\tau_{\tilde{u}}(x_{init}, \bar{t}(\tilde{u})) \in X_{goal}, \tag{2.122}$$

i.e., the final state of the trajectory is in the goal state set.

If Eq. (2.122) is replaced by

$$\|\tau_{\tilde{u}}(\bar{t}(\tilde{u})) - x_g\| < \epsilon_s \tag{2.123}$$

for some x_g in X_g and a real positive constant ϵ_s, the control $\tilde{u}$ is called a solution with *solution tolerance* ϵ_s. Furthermore, if the trajectory of $\tilde{u}$ from x_{init} has clearance w, then the solution is called to have *clearance* w. Given an *approximation tolerance*

$$0 < \epsilon_p < 1 \tag{2.124}$$

and a solution $\tilde{u}$ with clearance w, a control $\tilde{u}'$ is called the ϵ_p-*approximation* of the solution $\tilde{u}$ if its trajectory from the initial state is violation-free,

$$\tau_{\tilde{u}'}(x_{init}, t) \in T^{\epsilon_p w}_{\tau_{\tilde{u}}} \tag{2.125}$$

for

$$t \in [0, \bar{t}(\tilde{u}')], \tag{2.126}$$

i.e., any state in trajectory $\tau_{\tilde{u}'}(x_{init}, \cdot)$ is in the $(\epsilon_p w)$-tube of trajectory $\tau_{\tilde{u}}(x_{init}, \cdot)$, and

$$\|\tau_{\tilde{u}}(x_{init}, \bar{t}(\tilde{u})) - \tau_{\tilde{u}'}(x_{init}, \bar{t}(\tilde{u}'))\| < \epsilon_p w, \tag{2.127}$$

i.e., the final state of trajectory $\tau_{\tilde{u}'}(x_{init}, \cdot)$ is in the $(\epsilon_p w)$-neighborhood of the final state of trajectory $\tau_{\tilde{u}}(x_{init}, \cdot)$.

2.4 Path Planning Problems

Even though path planning problems are not the main topic in this thesis, their description is still provided here to give a complete view about various planning problems. Since differential constraints are not considered in the path planning problem, the action model of the system includes the configuration space and its constraints. A general motion planning problem, $\mathcal{P}$, is simplified as a tuple,

$$(\mathcal{C}, \mathcal{C}_{obs}, q_{init}, Q_g), \tag{2.128}$$

in which $\mathcal{C}$ and $\mathcal{C}_{obs}$ are the n_c-dimensional configuration space and its obstacles, and q_{init} and Q_g are the initial configuration and goal configuration set.

The closed set

$$\mathcal{C}_{obs} \subset \mathcal{C} \tag{2.129}$$

encodes the collision avoidance constraints from the work environment and configuration inequality constraints on the system. It is defined as

$$\mathcal{C}_{obs} = \mathcal{C}_{col} \cup \mathcal{C}_c. \tag{2.130}$$

The set

$$\mathcal{C}_{col} = \{q \in \mathcal{C} \mid T_G(q, G_r) \cap G_w \neq \emptyset\} \tag{2.131}$$

includes all configurations that cause the collision between the robot and work environment. The set

$$\mathcal{C}_c = \bigcup_{l=1}^{l_c} \{q \in \mathcal{C} \mid g_i^l(q) \not> 0\} \tag{2.132}$$

includes all configurations that violate the configuration inequality constraints from the system. Similarly, an open *collision-free* set, denoted as $\mathcal{C}_{free}$, is defined as

$$\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obs}. \tag{2.133}$$

The *initial configuration* and *goal configuration set* are respectively

$$q_{init} \in \mathcal{C}_{free} \tag{2.134}$$

and

$$C_{goal} \subset \mathcal{C}_{free}. \tag{2.135}$$

A *path*, denoted as p, in the configuration space is defined as

$$p : [0, 1] \rightarrow \mathcal{C}. \tag{2.136}$$

A collision-free and continuous path p is a solution if the path starts from the initial configuration and stops in the goal configuration set.

2.5 Nonholonomic Planning Problems

Since first-order nonholonomic constraints of system are considered, the action model of the nonholonomic planning problem includes the configuration space, kinematics, and their associated constraints. A general motion planning problem, $\mathcal{P}$, is simplified as a tuple,

$$(\mathcal{C}, \mathcal{C}_{obs}, U, \bar{\mathcal{U}}, f, q_{init}, Q_g), \tag{2.137}$$

in which $\mathcal{C}$ and $\mathcal{C}_{obs}$ are the configuration space and its obstacles, U and $\bar{\mathcal{U}}$ are the input space and control space generator set, q_{init} is the initial configuration, and Q_g is the goal configuration set.

The set $\mathcal{C}_{obs}$ and $\mathcal{C}_{free}$ are defined similarly as those for the path planning problem. The motion equation f of the system only encodes the kinematics, i.e.,

$$\begin{aligned} \dot{q} &= f(q,u) \\ &= f_q(q,u) \\ &= \sum_{l=1}^{n_v} u_l \mathbf{x}_l(q) \end{aligned} \tag{2.138}$$

in which u_l is the signed magnitude of the vector field $\mathbf{x}_l$, that is, the velocity of the system is directly controlled by its inputs.

The input space U includes all possible inputs and is defined as a rectangular subset of $\mathbb{R}^{n_v}$:

$$U = [u_1^{\min}, u_1^{\max}] \times [u_2^{\min}, u_2^{\max}] \times \cdots \times [u_{n_v}^{\min}, u_{n_v}^{\max}], \tag{2.139}$$

in which $n_v \leq n_c$, and $u_i^{\min}$ and $u_i^{\max}$ are the lower and upper bounds on the signed magnitude of the i-th vector field, and each input u satisfies l_v first-order inequality constraints

$$h_i^l(q,u) = h_i^l(q,v) > 0 \tag{2.140}$$

for $l = 1, 2, \cdots$, and l_v. The input space U is also equipped with a norm $\|\cdot\|$.

The control space $\mathcal{U}$ is similar to that for the general motion planning problem, except that a control for the nonholonomic system is a function from a time interval to the n_v-dimensional input space. A trajectory of control $\tilde{u}$ from configuration q_i is obtained by integrating the motion equation with $\tilde{u}$ from configuration q_i. Since the motion equation encodes only the kinematics, the trajectory of nonholonomic system is a function from a time interval to the configuration space.

The initial configuration and goal configuration set are respectively

$$q_{init} \in \mathcal{C}_{free} \tag{2.141}$$

and

$$Q_{goal} \subset \mathcal{C}_{free}. \tag{2.142}$$

A control $\tilde{u}$ is an exact solution if the value of control $\tilde{u}$ at any moment t satisfies input constraints, i.e.,

$$h_i^l(q, \tilde{u}(t)) > 0 \tag{2.143}$$

for $l = 1, 2, \cdots, l_v$ and any t in $[0, \bar{t}(\tilde{u})]$, the trajectory of $\tilde{u}$ from q_{init} is collision-free, i.e.,

$$\tau_{\tilde{u}}(q_{init}, t) \in \mathcal{C}_{free} \tag{2.144}$$

for any t in $[0, \bar{t}(\tilde{u})]$, and the trajectory stops in the goal configuration set, i.e.,

$$\tau_{\tilde{u}}(q_{init}, \bar{t}(\tilde{u})) \in Q_{goal}. \tag{2.145}$$

From the above problem description, it can be seen that the nonholonomic planning problem have the same representation as the general motion planning problem except that the state is replaced by the configuration and motion equation encodes kinematics instead of dynamics. From the perspective of the sampling-based algorithm, there is no difference between the description of the general motion planning problem and that of the nonholonomic planning problem. Both problems together will be called *MPD problems*, which have the following representation

$$\mathcal{P} = (X, X_{obs}, U, \bar{\mathcal{U}}, f, x_{init}, X_g), \tag{2.146}$$

but keep in mind that if it is a nonholonomic planning problem, the state is actually the configuration, and the motion equation encodes only the kinematics.

2.6 The Reachability Graph: an Intrinsic Graph Representation of MPD Problems

In the above sections, a representation of the problem is provided for the MPD algorithms. In this section, the problem is represented by the reachability graph, which

will be used in the resolution completeness analysis in Chapter 4.

The reachability graph is a directed graph which encodes all possible violation-free trajectories from the initial state for a given problem $\mathcal{P}$. It simply exists once $\mathcal{P}$ is defined and is *not* constructed by an algorithm. The reachability graph will serve as an important frame of reference for comparing the search graph generated by an algorithm. Every node in the reachability graph represents a *reachable state* from state x_{init}, for which there is a violation-free trajectory of a control from the initial state to the reachable state. The set of states of all nodes in the reachability graph is called the *reachable set* from state x_{init}, denoted as $\mathcal{R}_\infty(x_{init})$, which includes all reachable states from state x_{init}. Each directed edge corresponds to a violation-free trajectory of a control in the control space generator set. The trajectory starts from the state of the source node and stops at the state of the target node for that edge. Using the reachability graph instead of the reachable set to characterize the problem, a complete representation of the given MPD problem is provided, which include both the trajectories and reachable states. An example is shown in Figure 2.8, in which each dot represents a reachable state for a given problem and curves between dots represent trajectories associated with edges in the graph. Note that many reachable states and trajectories between reachable states are omitted.

To formally define the reachability graph, the set of *reachable states at stage k* from x_{init}, denoted as $R_k(x_{init})$, is introduced by induction. First,

$$R_0(x_{init}) = \{x_{init}\}. \tag{2.147}$$

At stage k,

$$R_k(x_{init}) = \{x \mid x = \tilde{f}(x', \tilde{u}),\ x' \in R_{k-1}(x_{init}),\ \tilde{u} \in \bar{\mathcal{U}}_{vf}(x')\}, \tag{2.148}$$

in which

$$\bar{\mathcal{U}}_{vf}(x) \subseteq \bar{\mathcal{U}} \tag{2.149}$$

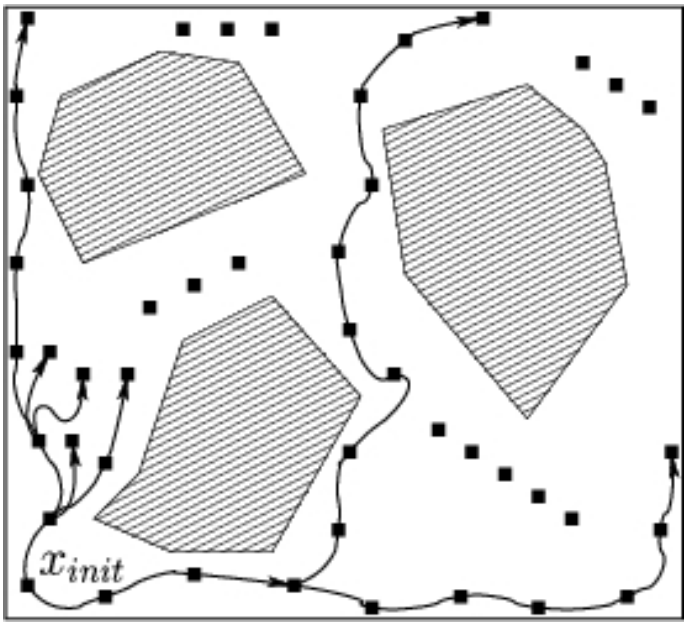


Figure 2.8: A reachability graph

denotes the set of controls that generate violation-free trajectories from x, and $\tilde{f}$ is the *discrete motion equation* defined for all $x_0 \in X$ and $\tilde{u} \in \mathcal{U}$ as

$$x' = \tilde{f}(x_0, \tilde{u}) = x_0 + \int_0^{\tilde{t}(\tilde{u})} f(x(s), \tilde{u}(s))\mathrm{d}s \tag{2.150}$$

The reachable set from state x_{init} is

$$\mathcal{R}_\infty(x_{init}) = \bigcup_{k=0}^{\infty} R_k(x_{init}). \tag{2.151}$$

Similarly, a *backward reachable set* from state x, denoted as $\mathcal{R}_\infty^-(x)$, is defined. For each state x' in $\mathcal{R}_\infty^-(x)$, there exists a violation-free trajectory of a control from state x' to state x. This concept will be used later to describe the sampling set for the state space sampling in Section 3.3.1.

Usually, the number of states in $\mathcal{R}_\infty(x_{init})$ is infinite because $\bar{\mathcal{U}}$ is infinite. Intuitively, when $\bar{\mathcal{U}}$ is finite as in some problems [53], it seems that $\mathcal{R}_\infty(x_{init})$ should be finite. However, it is shown in [53; 54] that $\mathcal{R}_\infty(x_{init})$ could be infinite for a class of discrete-time chained-form systems, even with a finite control space generator set. In the following part, $\bar{\mathcal{U}}$ is always assumed to be uncountable for conciseness. However, the analysis will also be applicable for problems with finite $\bar{\mathcal{U}}$.

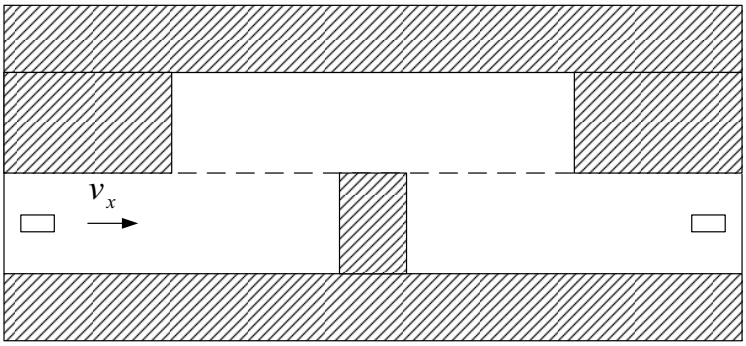


Figure 2.9: A lane change problem

The *reachability graph*, $\mathcal{G}\langle\mathcal{N},\mathcal{E}\rangle$, is defined as a directed graph. Every node n in $\mathcal{N}$ corresponds to a unique reachable state $x(n)$ in $\mathcal{R}_\infty(x_{init})$, and every directed edge $e(n_s, n_e)$ in $\mathcal{E}$ from node n_s to node n_e corresponds to a control $\tilde{u}(e)$ in $\bar{\mathcal{U}}_{vf}(x(n_s))$ with

$$x(n_e) = \tilde{f}(x(n_s), \tilde{u}(e)). \tag{2.152}$$

Note that the reachability graph is defined as a general graph, which is a set of nodes and a binary relation between nodes. Even though most graphs in computer science have a finite set of nodes, it is possible to define the reachability graph with an uncountable set of nodes.

2.7 An Example of the MPD Problem

The problem is shown in Fig. 2.9, in which a car with dynamics is required to do a lane change maneuver on a two lane road with 60mph constant forward speed in a 305m stretch of road, the shaded areas are obstacles to keep the car in the road and complete the lane change.

The state of the car in Fig. 2.10 is

$$[v_y, \omega, x, y, \theta]^T, \tag{2.153}$$

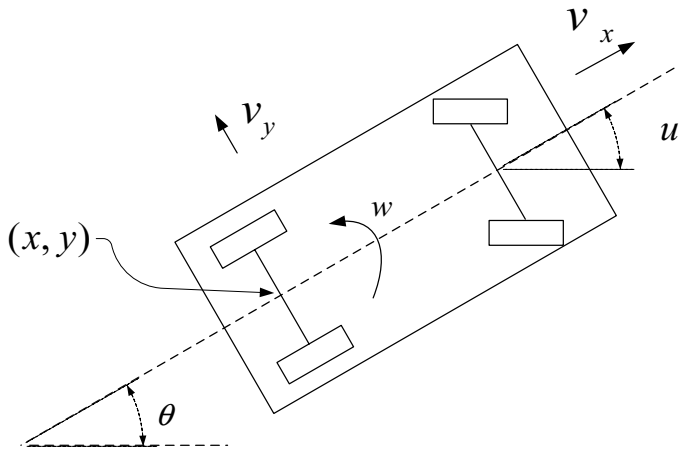


Figure 2.10: The sketch of the car with dynamics

in which

$$x \in [0, 800], \tag{2.154}$$

$$y \in [-800, -450], \tag{2.155}$$

and

$$\theta \in [-\pi, \pi] \tag{2.156}$$

represent the position and orientation,

$$\omega \in [-5, 5] \tag{2.157}$$

is the angular velocity,

$$v_y \in [-50, 50] \tag{2.158}$$

is the translational velocity perpendicular to the forward direction. The input to the system is the steering angle, denoted as u. The input space is $[-0.6, 0.6]$. The control space generator set includes all continuous functions from a time interval to the input space with control duration range be $[0, 2]$. This generator set is continuous

and uncountable. The motion equation is

$$\begin{cases} \dot{v}_y &= -v_x\omega + (f_{yf} + f_{yr})/M \\ \dot{\omega} &= (f_{yf}a - f_{yr}b)/I \\ \dot{x} &= v_x\cos(\theta) - v_y\sin(\theta) \\ \dot{y} &= v_x\sin(\theta) + v_y\cos(\theta) \\ \dot{\theta} &= \omega, \end{cases} \tag{2.159}$$

in which a and b are respectively the distance from the front and rear axles to the car mass center,

$$f_{yf} = -C_f((v_y + a\omega)/v_x - u) \tag{2.160}$$

and

$$f_{yr} = -C_r(v_y - b\omega)/v_x \tag{2.161}$$

are forces acting on front and rear tires along the direction perpendicular to the forward direction, C_f and C_r are constant coefficients of f_{yf} and f_{yr}, v_x is the constant forward velocity, and M and I are the mass and inertia.

The initial configuration and goal configuration are respectively shown as small rectangle boxes in left and right side of Fig. 2.9. The car starts and stops with v_y and ω be zero. The task is to design a control, which will drive the car from the initial state to the goal state without colliding obstacles, i.e., the car completes the lane change.

Chapter 3

A Sampling-Based Planning Framework

Sampling-based planning with differential constraints have been extensively applied to solve MPD problems, whose action models include differential constraints. This section presents an algorithm template that unifies numerous sampling-based planning under differential constraints. After the algorithm description in this chapter, it would be natural to discuss resolution completeness of these algorithms in Chapter 4, and algorithm designs in Chapters 5 and 6.

In the first section, a template for the algorithms will be given. In the second section, several sampling-based algorithms are described in the template. Sampling in the state space and control space will be specifically described and characterized in the last section.

3.1 A Template for Sampling-Based Planning

Given a problem $\mathcal{P}$, a search graph is iteratively built by sampling-based algorithms using sampled controls and states to construct solutions. The *search graph* is repre-

sented in the computer as a directed graph, $G\langle N, E\rangle$. A node

$$n \in N \tag{3.1}$$

is associated with a state $x(n)$ in X. The *reached set*, denoted as X_G, is defined as

$$X_G = \bigcup_{n \in N} \{x(n)\}, \tag{3.2}$$

which include all states reached by the algorithm. A directed edge

$$e(n_i, n_e) \in E \tag{3.3}$$

from node n_i to node n_e is associated with a control $\tilde{u}(e)$ and its trajectory $\tau_{\tilde{u}(e)}(x(n_i), \cdot)$ from state $x(n_i)$. Note that in some planners, such as bi-directional planners, the trajectory of control $\tilde{u}(e)$ could also be calculated by integrating backward-in-time from state $x(n_e)$. To be concise, the trajectory of an edge will always be from state $x(n_i)$ of the source node except that it is specifically mentioned to be calculated from $x(n_e)$.

Assume that a path p_G in the search graph consists of a sequence of edges

$$\{e_1(n_1, n_2), e_2(n_2, n_3), \cdots, e_k(n_k, n_{k+1})\}, \tag{3.4}$$

the *control of the path* is defined as

$$\tilde{u} = \tilde{u}(e_1) \circ \tilde{u}(e_2) \circ \cdots \circ \tilde{u}(e_k), \tag{3.5}$$

and the *trajectory of the path*, denoted as $\hat{\tau}$, is defined as

$$\hat{\tau} = \tau_{\tilde{u}(e_1)}(x(n_1), \cdot) \circ \tau_{\tilde{u}(e_2)}(x(n_2), \cdot) \circ \cdots \circ \tau_{\tilde{u}(e_k)}(x(n_k), \cdot), \tag{3.6}$$

in which $\tau_{\tilde{u}(e_i)}(x(n_i), \cdot)$ is the trajectory associated with edge e_i. The trajectory $\hat{\tau}$ of the path could be different from the trajectory $\tau_{\tilde{u}}(x_{init}, \cdot)$ of the control of the path from the initial state. The trajectory $\hat{\tau}$ is associated with the search graph and could be a discontinuous function from a time interval to the state space (an example is shown as thick curves in the upper picture in Fig. 3.1.). Trajectory $\tau_{\tilde{u}}$ is associated

with the reachability graph and normally is a continuous function from a time interval to the state space since it is obtained by integration. The difference between $\hat{\tau}$ and $\tau_{\tilde{u}}$ is one of the reasons that affect resolution completeness of sampling-based MPD algorithms. It will be described in details in Chapter 4. From Eq. (2.113), it can be seen that

$$\hat{\tau}(t) = \tau_{\tilde{u}}(x_{init}, t) \tag{3.7}$$

for

$$t \in [0, \bar{t}(\tilde{u})] \tag{3.8}$$

when

$$x(n_1) = x_{init}, \tag{3.9}$$

and

$$\tau_{\tilde{u}_i}(x(n_i), \bar{t}(\tilde{u}_i)) = x(n_{i+1}), \tag{3.10}$$

in which

$$\tilde{u}_i = \tilde{u}(e_i) \tag{3.11}$$

and $i = 1, 2, \cdots, k-1$.

A path is called a *solution path* if the control of the path is a solution. An example of the search graph and solution path is shown in Fig. 3.1, in which the middle picture contains an intermediate search graph that will be adjusted into the search graph in the lower picture, the upper picture shows the states and trajectories associated with nodes and edges in the search graph, the thick lines in the search graph show the solution path, and the thick curves in the upper picture are the discontinuous trajectory of the solution path.

Given an *initialization procedure*, *node selection*, *local planner*, *updating policy*, *solution checking policy*, and *termination condition*, the template of sampling-based planning algorithms is briefly described here, and then further explanation follows with specific examples.

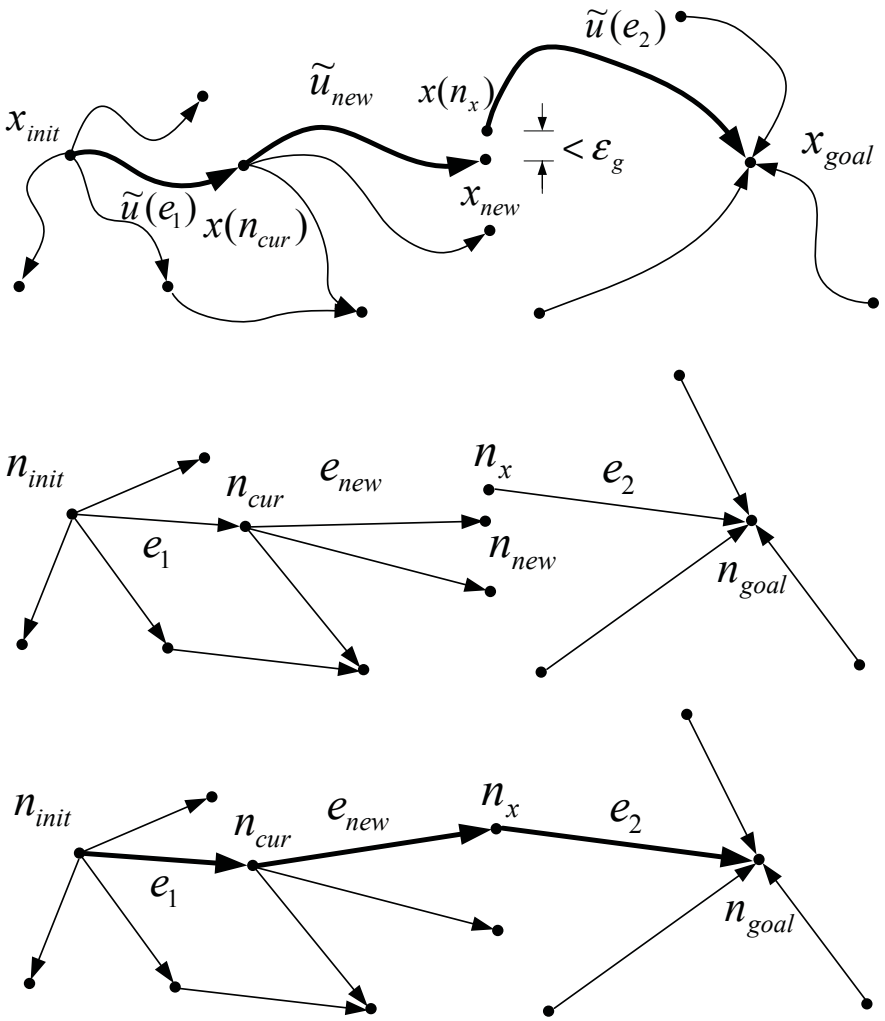


Figure 3.1: An example of the search graph, solution path, and generation of a new edge

1. **Initialize Search Graph:** Use the given initialization procedure to build a search graph with one, two, or multiple nodes and no edges. One of these nodes is called the *initial state node*, denoted as n_{init}, which is associated with the initial state x_{init} or a state in its neighborhood. In Fig. 3.1, the search graph is initialized with two nodes n_{init} and n_{goal}, which are respectively associated with states x_{init} and x_{goal}.

2. **Select Node:** Use the given node selection to choose a node n_{cur} in N. In Fig. 3.1, node n_{cur} is selected.

3. **Generate Trajectory Segment:** Use the local planner to sample a control $\tilde{u}_{new}$ from $\mathcal{U}$ and generate its trajectory from $x(n_{cur})$ to state x_{new}. In the upper picture of Fig. 3.1, a local planner samples the control $\tilde{u}_{new}$ and generates its trajectory from $x(n_{cur})$ to state x_{new}.

4. **Update Search Graph:** Use the updating policy to update G from two aspects: 1) check whether a new node n_{new} and new edge e_{new} should be inserted into the search graph. If yes, insert the node and edge. The new edge is associated with the sampled control $\tilde{u}_{new}$; 2) adjust the search graph with the new node and edge, which includes checking whether two subgraphs could be connected or whether the new node should be identified as a *goal state node*. The goal state node is associated with a goal state or a state in its neighborhood. In Fig. 3.1, a new edge e_{new} from node n_{cur} to n_{new} is inserted in the middle picture. The state of node n_{new} is x_{new}. In the lower picture, since the distance between states of nodes n_x and n_{new} is less than some given tolerance ϵ_g, the two subgraphs are connected by unifying two nodes as one node n_x.

5. **Check for Solution:** The solution checking policy determines whether a solution path exists. If a solution path is found, the control of the path is returned as the solution. In Fig. 3.1, a solution path in the lower picture consists of the following edges

$$\{e_1, e_{new}, e_2\}, \tag{3.12}$$

and the returned control is

$$\tilde{u} = \tilde{u}(e_1) \circ \tilde{u}(e_{new}) \circ \tilde{u}(e_2). \tag{3.13}$$

6. **Check Termination Condition:** Iterate until the given termination condition is satisfied.

Initialize search graph Given a problem, the initialization procedure generates a set of nodes with no edges for the search graph. Each node is associated with a state. In a single-tree approach, such as the planner in [35], only the initial state node n_{init} associated with x_{init} exists in N. In a bi-directional approach, such as the bi-directional RRT-based planner in [55], a goal state node n_{goal} associated with a goal state x_{goal} is also included in N. For planners in [22], a state in the ϵ_r neighborhood of x_{init} is associated with n_{init}. One could even place thousands of initial nodes in N, as in the case of initializing a probabilistic roadmap (PRM) [14; 56] with uniform random samples from X_{free}, and the methods using such initialization are called PRM-based methods.

Select node The node selection selects a node from the search graph by giving nodes different priorities and selecting the node with the highest priority, which is similar in some ways to the search queue prioritization in classical AI search. If Dijkstra's algorithm is used, as in [35], then its node selection selects a node that has untried controls and the shortest distance to x_{init}. Other possibilities are depth-first, breadth-first [22; 43], or A^* [57]. In the case of an RRT-based planner [13], a random state x_{rand} is generated in X, and then the node whose state is the nearest (with respect to a distance function on X) to state x_{rand} over all nodes in G is returned. Numerous other possibilities exist based on other algorithms (e.g., [22; 36–38; 44]).

Generate trajectory segment Step 3 is implemented by a *local planner*, which may be considered as a separate component that samples a control $\tilde{u}_{new}$ from $\mathcal{U}$ that evolves the system from $x(n_{cur})$ to some state x_{new}.

In some planners [38; 45; 50; 58–60], state x_{new} is given to their local planners as a parameter by a state space sampling method. These local planners are referred as *connecting* local planners. Some of these local planners use steering methods which do not consider constraints from the work environment, and in other local planners

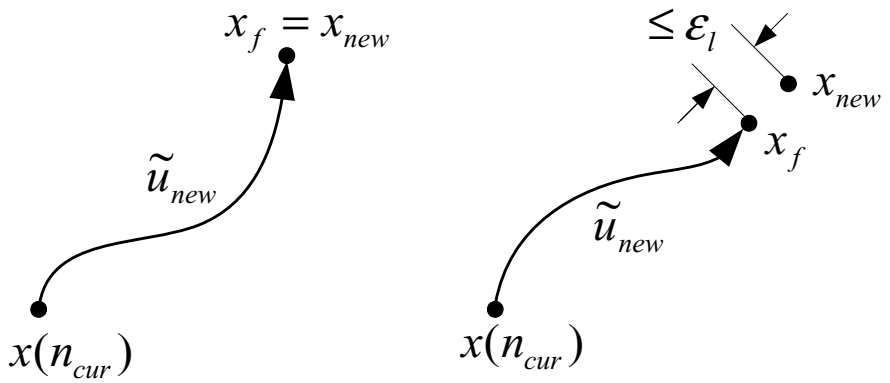


Figure 3.2: The trajectories generated from two types of connecting local planners

(e.g., [58; 59]) constraints from the work environment are considered. The trajectories generated by these local planners may either: 1) succeed in exactly reaching x_{new}, 2) succeed in reaching a given neighborhood of x_{new}, or 3) fail to reach sufficiently close to x_{new}, in which case another node must be selected in Step 2. For the second condition, a real positive constant ϵ_l is usually specified. The final state x_f of the generated trajectory must satisfy

$$\|x_{new} - x_f\| \leq \epsilon_l \tag{3.14}$$

to report success in reaching x_{new}. If a connecting local planner is permitted to succeed under condition 2, then it is called an *approximate connecting local planner* with tolerance ϵ_l; otherwise, it is called *exact* if it only succeeds under condition 1. Examples of successfully generated trajectories of two types of connecting local planners are shown in Fig. 3.2, in which the local planners try to sample a control whose trajectory is from $x(n_{cur})$ to a given state x_{new}, and the pictures from the left to the right show the trajectories generated by an exact connecting local planner and approximate connecting local planner.

Other local planners are called *non-connecting* local planners, which usually do not have a given target state x_{new}. They just sample a control $\tilde{u}_{new}$ from $\mathcal{U}$ and obtain x_{new} by integrating the motion equation with $\tilde{u}$ from $x(n_{cur})$.

Let

$$\tilde{\mathcal{U}}_s \subseteq \mathcal{U} \tag{3.15}$$

to denote the *sampled control set*, which includes all controls sampled by the local planner to construct G. A local planner normally only samples a specific class of controls, such as a piecewise constant acceleration [22; 43], a piecewise-constant control [13; 47], and non-constant controls [37; 38; 45; 48–50]. The *sampling control set*, denoted as $\tilde{\mathcal{U}}$, is the smallest generated control set that includes all possible controls that could be sampled by the local planner. For a local planner, a fixed and positive integer r is assumed in this thesis such that

$$\bar{\mathcal{U}}^{r-1} \not\subseteq \tilde{\mathcal{U}}_s \subseteq \bar{\mathcal{U}}^r = \tilde{\mathcal{U}}, \tag{3.16}$$

in which $\bar{\mathcal{U}}^{r-1}$ and $\bar{\mathcal{U}}^r$ are respectively $(r-1)$- and r-order generated control set (defined in Section 2.3). For connecting local planners, r is no less than 1. For example, the steering method for differential drive vehicles [28] needs at most five continuous controls to complete a steering; therefore, its r is 5. For non-connecting local planners, r is usually 1.

Since $\tilde{\mathcal{U}}$ is normally uncountable, an algorithm will not terminate in finite time if its local planner samples an infinite number of controls. Either a deterministic sampling [35; 36; 61] or a sampling with discretization would be used to obtain a finite sampled control set. These sampling is called *finite control space sampling*. In deterministic sampling, the finite sampled control set could be given before the algorithm starts. An example of sampling with implicit discretization is given in Section 3.3.2, in which a distance between controls in the sampled control set is maintained to achieve discretization in the control space.

If a control $\tilde{u}$ satisfies

$$\tilde{u} = \tilde{u}_1 \circ \tilde{u}_2 \circ \cdots \circ \tilde{u}_k, \tag{3.17}$$

in which $\tilde{u}_i$ is in $\tilde{\mathcal{U}}$, then $\tilde{u}$ is a *k-stage* control. If the control is a solution to problem

$\mathcal{P}$, then the solution is called a *k-stage* solution.

Update search graph The updating policies will first check whether the new trajectory from $x(n_{cur})$ is violation-free. If no, the new state will be discarded and the algorithm goes to the next step; otherwise, updating policies variate depending on whether state space discretization is used. If the states in set X_G are considered as sampled states, then the generation of states in set X_G is a state space sampling process. When a state space sampling only generates a finite X_G, it is called a *finite state space sampling*. State space discretization is one way to achieve a finite state space sampling[1].

When state space discretization is not used, there are the following possibilities. If state x_{new} is not in the set X_G, an edge associated with $\tilde{u}_{new}$ is always added to G from node n_{cur} to a new node associated with x_{new}. Otherwise, the edge is added from node n_{cur} to an existing node n_x because x_{new} equals $x(n_x)$. Examples of updating the search graph without state space discretization are shown in Fig. 3.3, in which n_x is an existing node in the search graph.

Without state space discretization, every x_{new} will be added to X_G if it is not in X_G. After the algorithm runs for an infinite number of iterations, an infinite set X_G could be generated since state x_{new} is normally not in the current X_G. For the nonconnecting local planners, state x_{new} is obtained by integrating a sampled control, which generally will not make x_{new} be in X_G. With connecting local planners, if state x_{new} is given by a random state space sampling in the state space X, x_{new} will be also added into X_G since the probability of generating state x_{new} in X_G is zero.

There are explicit and implicit ways of discretizing the state space. In the first way, the state space is explicitly partitioned into a finite set of *discretization sets*, in each of which at most one state in X_G is allowed such that only a finite number of

[1]There also exist other ways to achieve a finite reached set X_G, such as using a finite sampled control set and a finite search depth.

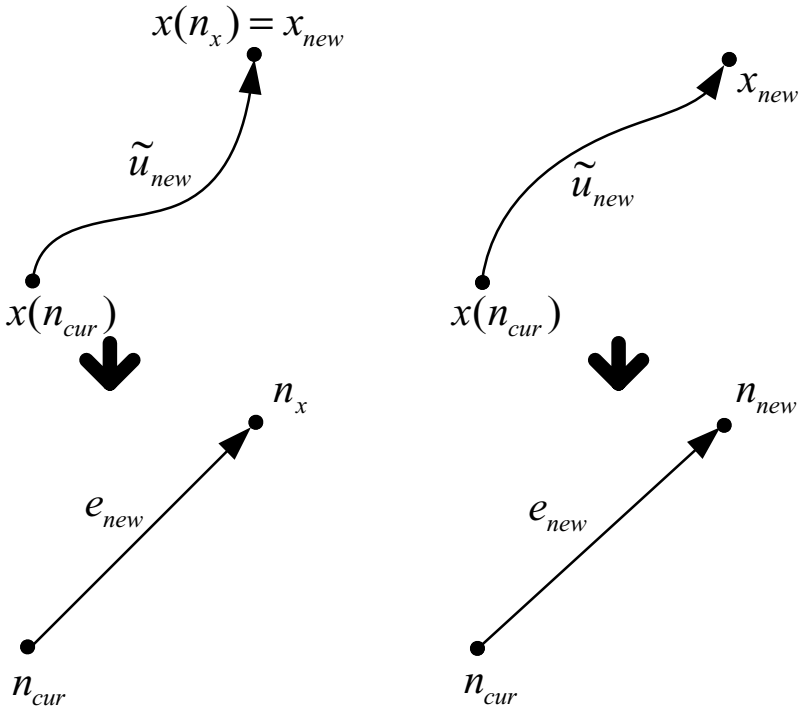


Figure 3.3: Updating the search graph without state space discretization

nodes could exist in G. If x_{new} is in a discretization set that does not contain the state of any node in G, then an edge from node n_{cur} to node n_{new} associated with state x_{new} is always added as shown in the bottom right picture in Fig. 3.4, in which the dashed lines discretize the state space. Otherwise, some planners, e.g., [35], discard x_{new} and no edge is added as shown in the bottom left picture in Fig. 3.4, while other planners, e.g., [62], discard x_{new}, but a new edge is inserted as shown in the bottom middle picture from n_{cur} to node n_x whose state is in the discretization set . Implicit state space discretization is achieved by requiring that the distance between states in X_G is no less than a given positive real constant. Based on whether state x_{new} is in the given neighborhood of a state in X_G, there are also three cases similar to those shown in Fig. 3.4. The explicit discretization is efficient but needs a large amount of memory space. The implicit discretization is slow but does not need extra memory space. In the method [35], the space is explicitly partitioned into a tiling of rectangular cells. In [62], the space is implicitly discretized.

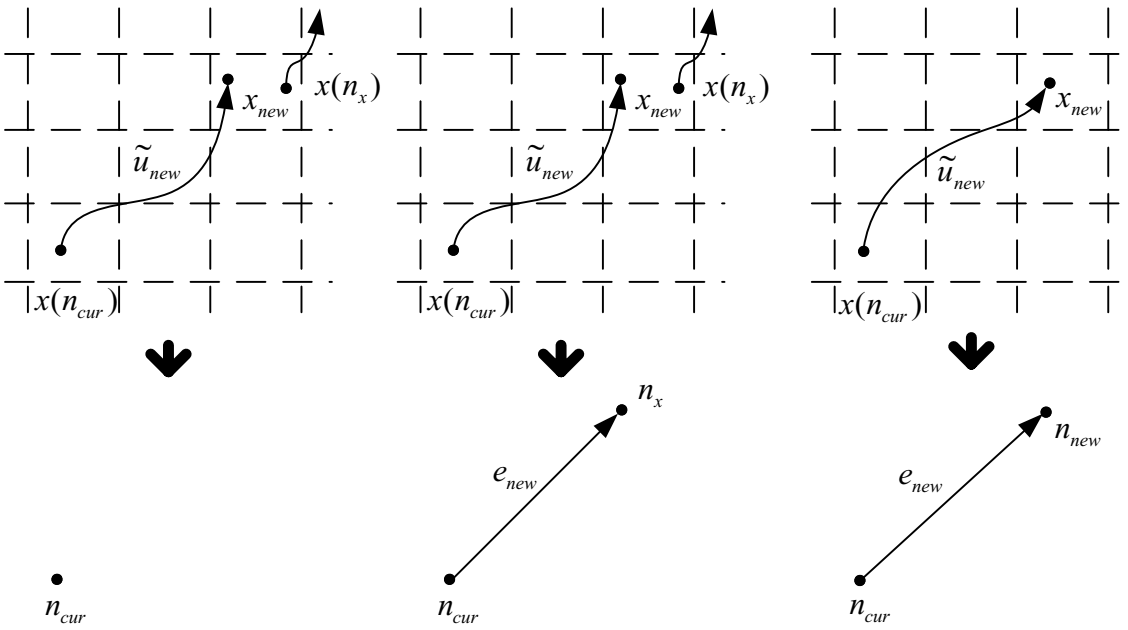


Figure 3.4: Updating the search graph with state space discretization

After a new node and edge are inserted, some policies will check whether two search subgraphs could be connected. In the bi-directional or PRM-based methods, the search graph consists of two or multiple subgraphs. If the distance between states of two nodes in two subgraphs is less than a given tolerance, then these two subgraphs are connected by unifying the two nodes. Some policies will check whether a goal state node could be added. If the state of the new node is in the given neighborhood of a goal state, then the new node is marked as a goal state node.

Check for solution A solution path is normally from the initial state node to a goal state node. Therefore, the solution checking policy will start a solution checking when two subgraphs are connected or a new goal state is marked. When two subgraphs are connected, a path from the initial state node in one subgraph could be connected to another path to a goal state node in the other subgraph as shown in Fig. 3.1. When the final node of a path is marked as a goal state node, the path could be a solution path if it also starts from the initial state node.

Check termination condition The termination condition tells whether the algorithm should stop. Some algorithms, such as [13], will stop after a given number of iterations. Some algorithm, such as [35; 36], will stop when all nodes in the search graph have been explored.

3.2 Descriptions of Sampling-Based Algorithms in the Template

In this section, three types of planners, which are the single-directional, bi-directional, and PRM-based planners, are described as specific examples of sampling-based planning with differential constraints. The single-directional search methods are from [13; 35]. The bi-directional one is from [13]. The PRM-based one is an extension from [13]. With the template in the above section, the algorithm description only needs the following components, which include the initialization procedure, node selection, local planner, solution checking policy, and termination condition.

3.2.1 The single-directional algorithms

The single-directional algorithm in [13] Given a problem, a distance function

$$\rho : X \times X \rightarrow \mathbb{R}^+ \cup \{0\}, \tag{3.18}$$

a finite sampled control set with m controls

$$\tilde{\mathcal{U}}_s = \{\tilde{u}_1, \tilde{u}_2, \cdots, \tilde{u}_m\}, \tag{3.19}$$

and a gap tolerance ϵ_g, RRT is a tree-like search graph, denoted as $\mathcal{T}$, which could be incrementally built to quickly explore the search space and construct solutions.

1. **Initialization procedure** It initializes the tree $\mathcal{T}$ with only one node n_{init} whose state is x_{init}.

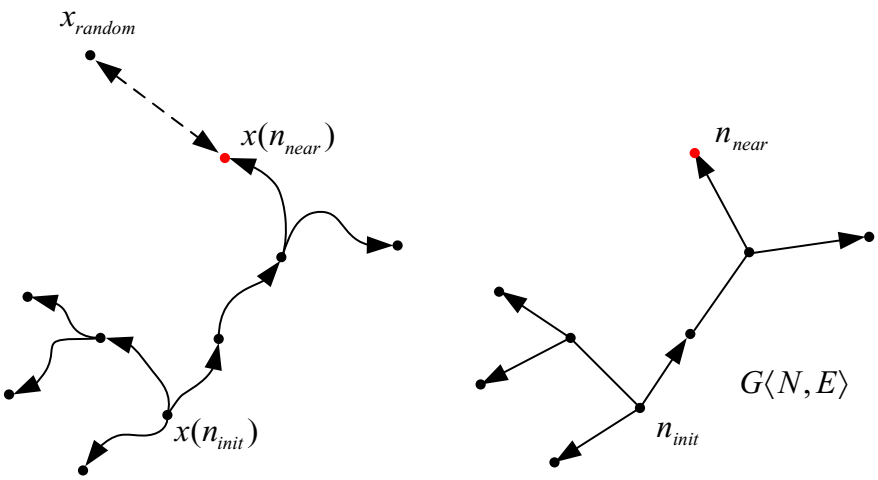


Figure 3.5: The node selection for RRT-based planners

2. **Node selection** As shown in Fig. 3.5, it chooses a node n_{near} in $\mathcal{T}$, whose state $x(n_{near})$ is the closest to a randomly generated state x_{random} with respect to the given distance function over states of all nodes in the search graph.

3. **Local planner** A nonconnecting local planner is used. As shown in Fig. 3.6, it will choose a control

$$\tilde{u}_{new} = \tilde{u}_k \tag{3.20}$$

from the given sampled control set $\tilde{\mathcal{U}}_s$ to generate a trajectory from $x(n_{near})$ to state

$$x_{new} = x_k, \tag{3.21}$$

which is the closest to x_{rand} over final states of trajectories of all controls in $\tilde{\mathcal{U}}_s$ from $x(n_{near})$.

4. **Updating policy** If the trajectory of $\tilde{u}_{new}$ from $x(n_{near})$ is violation-free, the policy always adds a new node n_{new} for the new state x_{new} and a new edge $e_{new}(n_{near}, n_{new})$; otherwise, do nothing. An example of a new edge is shown in Fig. 3.7. Note that the algorithm will not check whether state x_{new} is in set

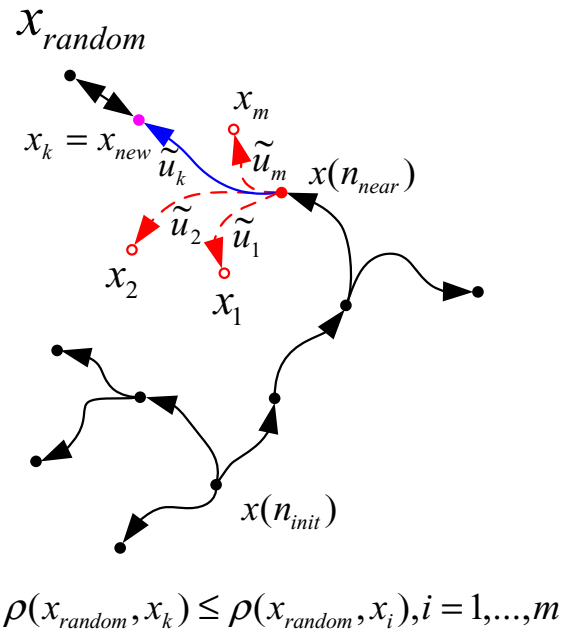


Figure 3.6: The local planner for RRT-based planners, in which the local planner samples $\tilde{u}_k$ from a finite set of m sampled controls

X_G or not.

Furthermore, if the state of the new node is in the ϵ_g neighborhood of a goal state x_{goal} in X_{goal}, then n_{new} is marked as a goal state node as shown in Fig. 3.8, in which the state $x(n_{new})$ of new node n_{new} is in the ϵ_g neighborhood of a goal state, and n_{new} is marked as a goal state node n_{goal}.

5. **Solution checking policy** When a goal state node is marked, a solution path exists and the control of the path is returned as a solution. An example of a solution path is shown in Fig. 3.8, in which the thick lines in the lower picture is the solution path, and the thick curves in the upper picture show the trajectory of the solution path.

6. **Termination Condition** The RRT-based planner in [13] will stop if a given number of iterations is reached or a solution path is found.

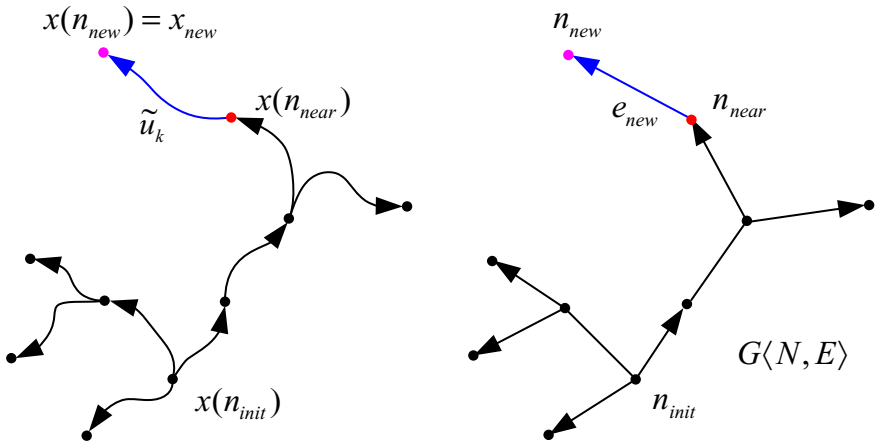


Figure 3.7: A new edge $e_{new}(n_{near}, n_{new})$ is added when the trajectory of $\tilde{u}_k$ from $x(n_{near})$ is violation-free

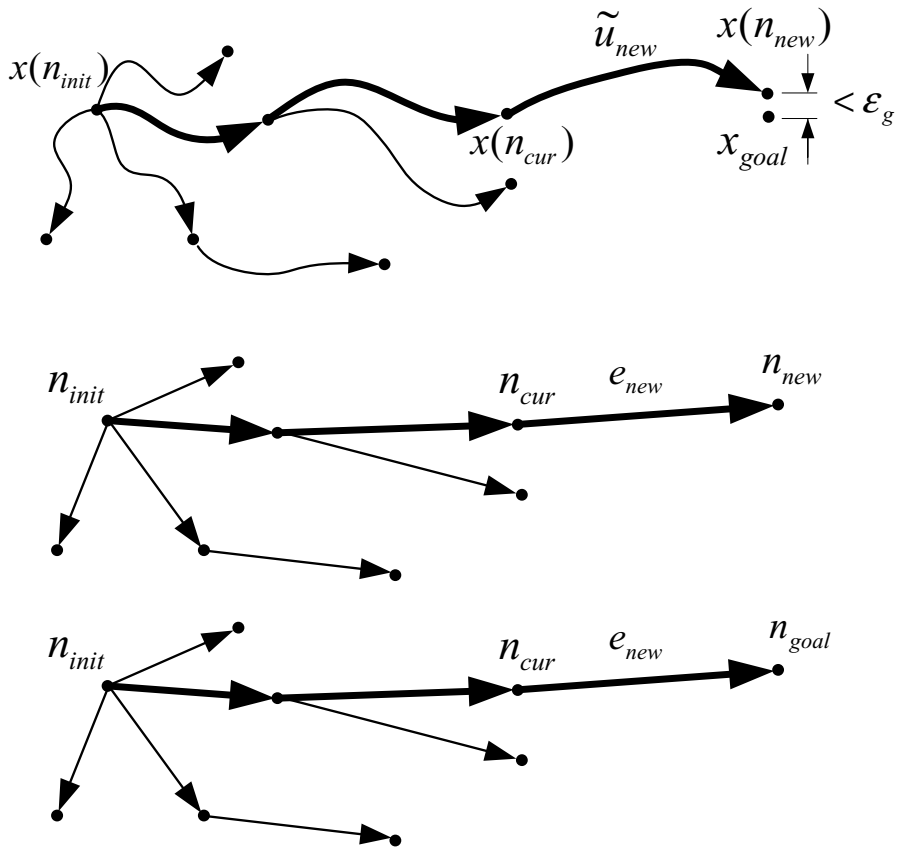


Figure 3.8: Finding a goal state node and solution path for the RRT-based single-directional method

The single-directional algorithm in [35] Even though the method in [35] was designed for nonholonomic path planning problems, it is also suitable for general motion planning problems by extending its search space from the configuration space to the state space. The search graph for this method is also a tree structure with the root node associated with the initial state. The algorithm overall works like the Dijkstra's algorithm. Its local planner also chooses sample controls from a finite sampled control set $\tilde{\mathcal{U}}_s$. Each sampled control is a constant control of a fixed duration δt.

To describe this algorithm exactly in the template, the nodes in the search graph are categorized as *explored*, *partially explored*, or *unexplored*. If none of the controls in the sampled control set have been applied on a node to generate trajectories from the state of the node, then the node is called unexplored. If all controls in the set $\tilde{\mathcal{U}}_s$ have been applied on a node, then the node is called explored. Otherwise, if only part of controls in the finite set have been applied on a node, then the node is called partially explored. Each node is also associated with a real value, which denotes the cost-to-come from the initial state to the state of the node.

Before the construction of the search graph, the algorithm discretizes the state space into smaller parallelepipeds of equal size. Specifically, for an n-dimensional state space, each dimension is divided into 2^R intervals of equal size such that there are 2^{nR} parallelepipeds. A parallelepiped is called *occupied* if the state of a node in the search graph is inside, or *empty* if otherwise.

1. **Initialization procedure** It initializes the tree with only one node n_{init} whose state is x_{init}. The node is marked as unexplored.

2. **Node selection** It checks each node in the search graph. If there exists a partially explored node, the node is selected. Otherwise, the unexplored node with the least cost is chosen. The selected node is denoted as n_{cur}.

3. **Local Planner** It chooses a control $\tilde{u}_{new}$ in the set $\tilde{\mathcal{U}}_s$, which has not been applied on node n_{cur}. A new trajectory of $\tilde{u}_{new}$ from state $x(n_{cur})$ is generated. The final state of the new trajectory is x_{new}.

4. **Updating policy** It first marks node n_{cur} to show the control $\tilde{u}_{new}$ has been applied. If $\tilde{u}_{new}$ is the first control applied, the node is marked as partially explored. If all controls have been applied, the selected node is marked as explored. Secondly, it checks whether the new trajectory is violation-free. If no, the algorithm goes to the next step; otherwise, it will check whether the parallelepiped where state x_{new} lives is occupied. If yes, x_{new} is discarded; otherwise, a new node n_{new} marked as unexplored and a new edge $e_{new}(n_{cur}, n_{new})$is added. The updating behavior is shown in the left and right pictures of Fig. 3.4. If the state of the new node is in a parallelepiped which contains a goal state, then the new node is marked as a goal state node.

5. **Solution checking policy** When a goal state node is marked, a solution path and its control is returned.

6. **Termination condition** The algorithm will terminate either when all nodes up to a given search depth K in the search graph have been marked as explored or a solution path is found.

3.2.2 A bi-directional algorithm

The bi-directional search methods normally initialize the search graphs as two disjointed subgraphs, which start respectively with the initial node and a goal state node. A solution path is found when states of two nodes in two subgraphs are close enough. Here, the bi-directional RRT-based planner in [13] is used as a specific example. Given a problem, a finite sampled control set $\tilde{\mathcal{U}}_s$, a distance function, and a tolerance ϵ_g, the components of the algorithm are given as follows.

1. **Initialization procedure** It initializes the search graph with two trees $\mathcal{T}_1$ and $\mathcal{T}_2$, which respectively have node n_{init} associated with state x_{init} and node n_{goal} associated with a goal state x_{goal}.

2. **Node selection** To balance the exploration of two trees, it takes turns to chooses a node n_{near} in $\mathcal{T}_1$ and $\mathcal{T}_2$, whose state $x(n_{near})$ is the closest to a state x_s with respect to the given distance function over states of all nodes in the respective tree. The state x_s could be a random state, or a new state generated from the other tree.

3. **Local planner** It will choose a control $\tilde{u}_{new}$ from the set $\tilde{\mathcal{U}}_s$ to generate a trajectory from $x(n_{near})$, whose final state x_{new} is the closest to state x_s of all final states of trajectories of controls in the finite set from $x(n_{near})$. Note that if node n_{near} is in the subgraph that is initialized with the goal state node, the trajectory is calculated by integrating backward-in-time from state $x(n_{near})$. An example is shown in Fig. 3.9, in which the thick curve in the upper picture represents the generated trajectory, and the thick line in the lower picture is the new edge from the search graph updating process.

4. **Updating policy** The policy will check whether the trajectory of $\tilde{u}_{new}$ from $x(n_{near})$ is violation-free. If no, state x_{new} is discarded and the algorithm goes to the next step; otherwise, a new node n_{new} associated with the new state x_{new} and a new edge e_{new} will be added. If node n_{near} is selected from $\mathcal{T}_2$ that contains the goal state node, the new edge e_{new} is from node n_{new} to node n_{near} as shown in Fig. 3.9; otherwise, the new edge e_{new} is from node n_{near} to node n_{new}.

   Since the search graph consists of two disjointed subgraphs, the policy will check whether the distance between the states of two new nodes from two trees is less than the given tolerance ϵ_g. If yes as shown in Fig. 3.1, then two subgraphs are

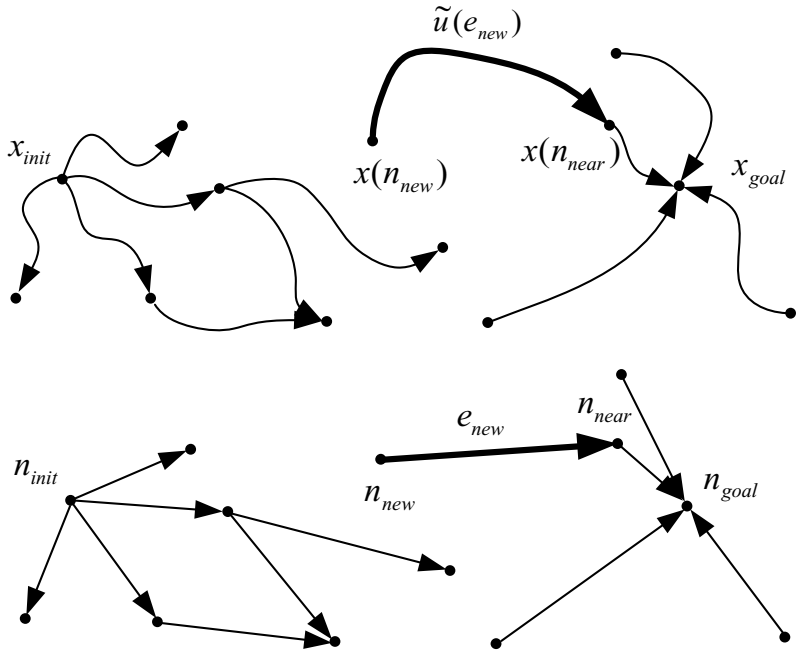


Figure 3.9: Backward-in-time trajectory generation and graph updating for the subgraph that contains a goal state node

connected by unifying nodes n_{new} and n_x as one node n_x.

5. **Solution checking policy** If two subgraphs are connected, the path from the initial state node in $\mathcal{T}_1$ and the path to the goal state node in $\mathcal{T}_2$ generate a solution path; otherwise, go to the next step.

6. **Termination Condition** The algorithm will stop if a given number of iterations is reached or a solution path is found.

3.2.3 A PRM-based search algorithm

PRM-based planning algorithms [14] was originally developed for path planning problems. The algorithm consists of the construction phase and query phase. In the construction phase, a roadmap, which is an undirected graph, is built incrementally with sample points in the configuration space to capture the connectivity of the collision-free configuration space. In each iteration, a sampled point tries to connect the neighboring points in the existing roadmap. In the query phase, the initial con-

figuration and goal configuration try to connect to nodes in the roadmap. If both initial configuration and goal configuration are connected to the roadmap and there is a path from the initial configuration to the goal configuration, then a solution is returned. With the constructed roadmap, the search in the query phase tends to have lower search depth and therefore faster query time. These algorithms have successfully solved many challenging path planning problems. It is desirable to extend the method for MPD problems. As shown in Section 3.2.2 that a search graph could be built with two disjointed subgraphs in the bi-directional algorithm, it is natural to construct a search graph with multiple disjointed subgraphs, which will obtain a PRM-based planning method with differential constraints. In the following, a PRM-based planning method with differential constraints is described by extending RRT-based planners. Both the construction and query phases use the following components with minor differences.

1. **Initialization procedure** In the construction phase, k_1 new disjointed subgraphs are added to the search graph. Each new subgraph is initialized with only one node associated with a sample state, which is not in the given neighborhood of the state of any node in the current search graph. In the query phase, two new disjointed subgraphs are initialized. One only has a node associated with the initial state, and the other one only has a node associated with the goal state.

2. **Node selection** In the construction phase, to balance the exploration of different subgraphs, it takes turns to choose a node n_{near} in each subgraphs, whose state $x(n_{near})$ is the closest to a state x_s with respect to the given distance function over states of all nodes from the same subgraph. The state x_s could be a random state, or a new state generated from the other subgraphs. In the query phase, it takes turns to choose a node n_{near} from subgraphs with the initial and goal state nodes.

3. **Local planner** It will choose a control $\tilde{u}_{new}$ from a finite set of sampled controls to generate a new trajectory from $x(n_{near})$, whose final state x_{new} is the closest to x_s of all final states of trajectories of controls in the finite set from $x(n_{near})$.

4. **Updating policy** In both phases, the policy will first check whether the new trajectory of $\tilde{u}_{new}$ from $x(n_{near})$ is violation-free. If no, state x_{new} is discarded and the algorithm goes to the next step; otherwise, a new node n_{new} associated with the new state x_{new} and a new edge e_{new} will be added. The distance between the state of the new node to states of nodes in other subgraphs is further checked. If the distance is less than tolerance ϵ_g, then the two subgraphs are connected by unifying the two nodes.

   In the query phase, if the state of the new node is in the ϵ_g neighborhood of a goal state, the new node is marked as a goal state node.

5. **Solution checking policy** The solution checking only happens in the query phase. When two subgraphs are connected at a unified node or a new goal state node is marked, whether there is a solution path through the unified node or with the new goal state node as the final node will be checked. If there is a solution path, then its control is returned.

6. **Termination Condition** The construction phase will stop after a given number of iterations. The query phase will stop either when a solution path is found or a given number of iterations are reached.

3.3 Characterization of State Space Sampling and Control Space Sampling

To facilitate the resolution completeness analysis in Chapter 4, the sampling in the state and control space is described and characterized here. The objective is to provide

an *invariant* characterization which is independent of specific sampling-based MPD algorithms and their different runs, i.e, these characterizations will be invariant when the sampling is used in different algorithms and their different runs.

The characterization needs the concept of *dispersion* [63]. For a set B with a norm $\|\cdot\|$, a set A consists of sampled points from set B. The dispersion of set A with respect to set B is defined as

$$\sup_{b\in B}\inf_{a\in A}\|a-b\|. \tag{3.22}$$

Intuitively, it is the furthest away any state in set B could be from its nearest sampled point in set A.

3.3.1 State space sampling

The states of in the reached set X_G of a search graph could be considered as a sampled point in a *sampling state set*, denoted as $\tilde{X}_G$, which is the smallest state set that includes all possible X_G. The process of generating new state in X_G could be considered as a state space sampling from set $\tilde{X}_G$.

For a search graph which is initialized with nodes

$$\{n_1^+, n_2^+, \cdots, n_a^+, n_1^-, n_2^-, \cdots, n_b^-, n_1^\pm, n_2^\pm, \cdots, n_c^\pm\}, \tag{3.23}$$

in which n_i^+ means a subgraph will be built with forward-in-time integration from state $x(n_i^+)$, n_j^- means that a subgraph will be built with backward-in-time integration from state $x(n_j^-)$, and $n_k^\pm$ means that a subgraph will be built with both forward- and backward-in-time integration from state $x(n_k^\pm)$ described in Section 3.2.2, the sampling state set is defined as

$$\tilde{X}_G = \bigcup_{i=1}^{a}\mathcal{R}_\infty(x(n_i^+))\bigcup_{j=1}^{b}\mathcal{R}_\infty^-(x(n_j^-))\bigcup_{k=1}^{c}\mathcal{R}_\infty(x(n_k^\pm))\bigcup_{k=1}^{c}\mathcal{R}_\infty^-(x(n_k^\pm)), \tag{3.24}$$

i.e., the union of reachable sets from states $\{x(n_i^+)\}$ and $\{x(n_k^\pm)\}$ and the backward reachable set from states $\{x(n_j^-)\}$ and $\{x(n_k^\pm)\}$. For a single-directional MPD algo-

rithm with the initial state node associated with state x_{init}, its sampling state set is $\mathcal{R}_\infty(x_{init})$.

Since the reached set X_G and the sampling state set $\tilde{X}_G$ would change using the same sampling method in different algorithms or even different runs of the same algorithm, the state space sampling in $\tilde{X}_G$ should be characterized using the *dispersion bound of the state space sampling in* $\tilde{X}_G$, which is an upper bound on the dispersion of all resulting reached sets with respect to corresponding sampling state sets. In this way, the characterization will be independent of specific algorithm and their different runs. Since the sampling state set $\tilde{X}_G$ is normally unknown, it is impossible to evaluate the dispersion of X_G with respect to $\mathcal{R}_\infty(x_{init})$ and the associated dispersion bound. However, when state space discretization is used, the state space sampling in $\tilde{X}_G$ is actually characterized by the *dispersion bound of a state space sampling with discretization in* X, which is an upper bound on the dispersion of the *maximal sampled set*, denoted as X_s, with respect to X over all possible X_s. Each element of a maximal sampled set X_s is obtained through the sampling in X and no more state in X could be added into X_s while still satisfying the discretization rule. Even though the dispersion bound of the state space sampling in $\tilde{X}_G$ is different from that of the state space sampling with discretization in X, the second dispersion bound will provide sufficient information for the analysis in Chapter 4. To be concrete, two examples of such characterization are given as follows.

If an explicit state space discretization is used in Step 4 to prune states, there is at most one state in X_G in each discretization set. A state space sampling with discretization in X is obtained by sampling one state from each discretization set. The maximal sampled set X_s is constructed by sampling one state from each discretization set. Even though with the same state space discretization X_G would change from different algorithms or even from different runs of the same algorithm, such as the randomized algorithm, it is always a subset of some maximal sampled set X_s. An

invariant characterization will be the dispersion bound on the dispersion of all possible X_s with respect to the state space.

If a connecting local planner is used in Step 3, then a state space sampling with discretization in X is usually used to achieve the sampling in $\tilde{X}_G$. With an implicit discretization, a new sample state x_{new} will be added to X_G if it is successfully connected to a state in the current X_G and is not in a given neighborhood of a state in X_G. A maximal sampled set X_s is a set of states whose given neighborhoods cover the state space and whose distance between each other is no less than the given distance. The set X_s is finite since the state space is bounded. Any X_G is always a subset of some maximal sampled set X_s. The sampling is characterized by the dispersion bound on the dispersion of the finite set X_s with respect to the state space.

3.3.2 Control space sampling

In Step 3, a local planner is used to sample a control from the sampling control set $\tilde{\mathcal{U}}$. An invariant characterization for control space sampling will be limited to finite control space sampling and based on whether a connecting local planner is used.

When an exact connecting local planner is used, under the assumptions that motion equations have a unique solution and the local planner returns a unique solution given two states, the sampling in the control space is transformed into that in the state space. Given a chosen node n_{cur}, for every sampled state x_{new}, if the local planner successfully connects it from state $x(n_{cur})$, then a control is sampled. Discretization in the control space is also achieved by that in the state space. Thus, the control space sampling with discretization could be characterized by the dispersion bound of the state space sampling with discretization in X. Similar characterization could be extended for an approximate connecting local planner with tolerance ϵ_l. The approximate connecting local planner is modeled as an exact local planner plus a sampling process. For each sampled state x_{new}, a state x_s in the ϵ_l neighborhood of the state

x_{new} is sampled first. If state $x(n_{cur})$ could be connected to x_s by the associated exact local planner, then a control is sampled. The control space sampling with an approximate connecting local planner is characterized by the dispersion bound ϵ_d and tolerance ϵ_l.

When a non-connecting local planner is used, the sampling control set $\tilde{\mathcal{U}}$ equals the control space generator set $\bar{\mathcal{U}}$ and a control is sampled from $\bar{\mathcal{U}}$ in two steps. The first sampling in the duration interval $\mathcal{D} \setminus \{0\}$, called *time sampling*, obtains a duration t, which corresponds to a set of controls,

$$\bar{\mathcal{U}}_t = \{\tilde{u} \in \bar{\mathcal{U}} \mid \bar{t}(\tilde{u}) = t\}. \tag{3.25}$$

The second sampling in $\bar{\mathcal{U}}_t$, called *input space sampling*, obtains a control $\tilde{u}_s$. A finite sampled control set

$$\tilde{\mathcal{U}} = \{\tilde{u}_1, \tilde{u}_2, \cdots, \tilde{u}_l\} \subseteq \bar{\mathcal{U}}, \tag{3.26}$$

is characterized by two positive parameters ϵ_t and ϵ_u, which respectively for the first and second sampling. The first parameter ϵ_t, called a *dispersion bound of the time sampling*, is an upper bound on the dispersion of

$$T_s = \{t \mid t = \bar{t}(\tilde{u}_k),\ k = 1, 2, \cdots, l\} \tag{3.27}$$

with respect to $\mathcal{D}$ over all possible $\tilde{\mathcal{U}}$ obtained by the control space sampling. The second parameter ϵ_u, called a *dispersion bound of the input space sampling*, is a uniform upper bound on the dispersion of

$$\tilde{\mathcal{U}}_t = \{\tilde{u} \in \tilde{\mathcal{U}} \mid \bar{t}(\tilde{u}) = t\} \tag{3.28}$$

with respect to $\bar{\mathcal{U}}_t$ for all t in T_s over all possible $\tilde{\mathcal{U}}$ obtained by the control space sampling. Note that that the dispersion of the second sampling is defined with respect

to infinity norm[2], $\|\cdot\|_\infty$ on $\mathcal{U}$, which is defined as

$$\|\tilde{u}\|_\infty = \sup_{s\in[0,\bar{t}(\tilde{u})]} \|\tilde{u}(s)\|. \tag{3.29}$$

The distance between two controls $\tilde{u}$ and $\tilde{u}'$ is defined as

$$\|\tilde{u} - \tilde{u}'\| = \sup_{s\in[0,\min(\bar{t}(\tilde{u}),\bar{t}(\tilde{u}'))]} \|\tilde{u}(s) - \tilde{u}'(s)\|. \tag{3.30}$$

To construct a sampled control set with dispersion bounds ϵ_t and ϵ_u, an implicit discretization could be combined with a random sampling. For every newly sampled control $\tilde{u}$, if its duration t is not in the ϵ_t neighborhood of duration of any other control in $\tilde{\mathcal{U}}_s$ and its distance to any control in $\tilde{\mathcal{U}}_t \cap \tilde{\mathcal{U}}_s$ is larger than ϵ_u, it is added to $\tilde{\mathcal{U}}_s$. The construction will be done when no new control could added. An example of deterministic finite control space sampling is shown in Appendix A.3.2.

[2]The analysis could be adapted to norms of the form $\|\cdot\|^p = (\int_0^t \|\cdot\|^p \mathrm{d}s)^{\frac{1}{p}}$ for $p \geq 1$ with minor modification. However, the infinity norm is used for conciseness.

Chapter 4

Resolution Completeness Analysis

Completeness provides performance guarantee for algorithms. In finite time, a *complete* algorithm will either find an existing solution or report that no solution exists for a given problem. To provide completeness, an algorithm needs a finite and exact representation of the search space of a problem, which captures all information of the problem and could be exhaustively explored in finite time; otherwise, the algorithm might not be complete. When an exact and finite representation of the search space is unavailable or expensive to calculate, sampling techniques could be used to generate a set of sampling points to approximate this space and construct solutions. Sampling techniques have been used in many areas, such as numerical integration, motion planning, and computer graphics. To numerically integrate a real-valued function over a complicated closed set when analytical integration is not available, a set of sample points could be used to approximate the closed set and the function. In computer graphics, when there are too many geometric primitives for a model, volumetric visualization [64] could be used to represent the model by a set of points, or billboard clouds [65] could be used to model the geometry by a set of planes. In motion planning, the configuration space of the Piano Movers' Problem is only a continuous search space and the exact representations, such as cylindrical algebraic decomposition [10] and roadmap [11], of the free configuration space $\mathcal{C}_{free}$ are very

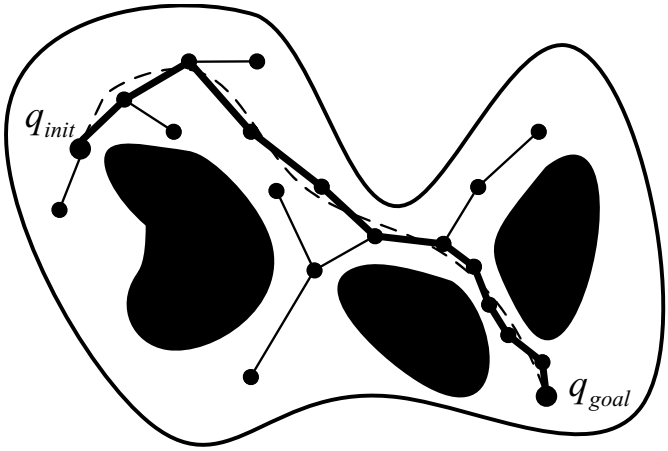


Figure 4.1: Intuition of resolution completeness of path planning algorithms

expensive. Sampling-based path planning algorithms solve this problem by iteratively building a search graph by sampling the continuous configuration space.

Sampling-based algorithms quickly find practical solutions at the cost of losing completeness since the problem representation is not exact. Only weaker completeness, such as *probabilistic completeness* or *resolution completeness*, could be achieved. Probabilistic completeness means that the probability for an algorithm to find an existing solution will approach one when the algorithm runs for ever. Resolution completeness means that an existing solution or its approximation will be found in finite time if resolution of sampling points is high enough. In the case of sampling-based path planning algorithms, a resolution complete algorithm will find an existing solution provided that resolution of sampled points in the configuration space is high enough [18]. The main idea of resolution completeness for path planning algorithms could be seen from Fig. 4.1, in which the dashed curve is a solution, the black dots are sampled configurations, line segments between two dots are edges of the search graph, and the thick lines represent another solution that is constructed by the sampling-based path planning algorithm. If resolution of the sampled points is high enough, then there always exists a sequence of sample points around the dashed path such that connecting these sample points will generate a solution.

For a given MPD problem $\mathcal{P}$, there are normally three continuous search spaces: time, the input space (the control space could be considered as Cartesian product of time and the input space), and the state space. A sampling-based planner for these problems will normally sample all three of the continuous spaces simultaneously. Therefore, it is essential to extend the resolution completeness concept to all three spaces. A sampling-based MPD planner is resolution complete if there exists a solution for a given problem, the algorithm will find it or its approximation in finite time when the sampling process and parameters for the three continuous spaces are appropriately chosen.

The key to resolution completeness of sampling-based MPD is to understand that the search graph described in Section 3.1, which is constructed by a sampling-based planning algorithm, could be only an approximation of a subgraph of a reachability graph described in Section 2.6, which is an intrinsic graph representation of a given MPD problem. The reachability graph is fixed once an MPD problem is given, as opposed to being computed by an algorithm. If there is a solution for the problem, there must exist a solution path in the reachability graph, whose trajectory from the initial state will be violation-free and stop at a goal state. A planning algorithm builds a search graph which ideally should incrementally reveal the reachability graph, and eventually converge to it. However, because of mismatches caused by sampling in the control space and state space, approximate local planners, numerical calculation, and other factors in the algorithm, the search graph might be only an approximation of a subgraph of the reachability graph (will be explained in details in Section 4.2.2) and completeness of sampling-based MPD algorithms will be affected. A resolution complete algorithm must find an existing solution or its approximation in finite time despite of these mismatches.

Mismatches could exist in the state space or control space, called *state mismatches* and *control mismatches*, respectively. State mismatches happen when 1) the trajec-

tory of an edge of the search graph does not exactly connect the states of two nodes of the edge; 2) the initial state node is not associated with the initial state; 3) a goal state node is not associated with a goal state; or 4) the states of initial nodes in the search graph are not reachable from the initial state. Control mismatches happen when the sampled control set $\tilde{\mathcal{U}}_s$ is just a subset of the sampling control set $\tilde{\mathcal{U}}$.

State mismatches destroy completeness by returning a wrong solution, preventing solutions from being detected, or failing to get sufficiently close to a goal state. Effects of a state mismatch are shown in Fig. 4.2, in which X_{obs} is the obstacle in the state space described in Section 2.3, Pictures (2) and (3) include the search graphs, thick lines in Picture (3) represent a returned solution path, Picture (1) includes the trajectories of the edges, thick curves in Picture (1) represent the trajectory of the solution path defined in Section 3.1, and Picture (4) shows the trajectory of the control of the returned solution path from the initial state. A state mismatch exists in the search graph in Picture (3) since the trajectory of edge e_{new} does not connect $x(n_{cur})$ and $x(n_x)$ as shown in Picture (1). The mismatch is induced by unifying nodes n_{new} and n_x in Picture (2) as node n_x in the process of updating the search graph. Even though the trajectory $\hat{\tau}$ of the solution path is violation-free as shown in Picture (1) of Fig. 4.2, the state mismatch in edge e_{new} causes that the returned control is not a correct solution since its trajectory from the initial state is not violation-free and its final state is not sufficiently close to the goal state x_{goal} as shown in Picture (4) of Fig. 4.2.

Control mismatches directly come from the usage of sampling in the control space. The sampled control set $\tilde{\mathcal{U}}_s$, whose elements are used to construct the search graph, is always a subset of the sampling control set $\tilde{\mathcal{U}}$, whose controls are used to construct the reachability graph. The direct result of control mismatches is that some solutions to the problem will not exist in the search graph. Only η-neighboring controls and trajectories (which are described in Section 2.3) of the solution and its

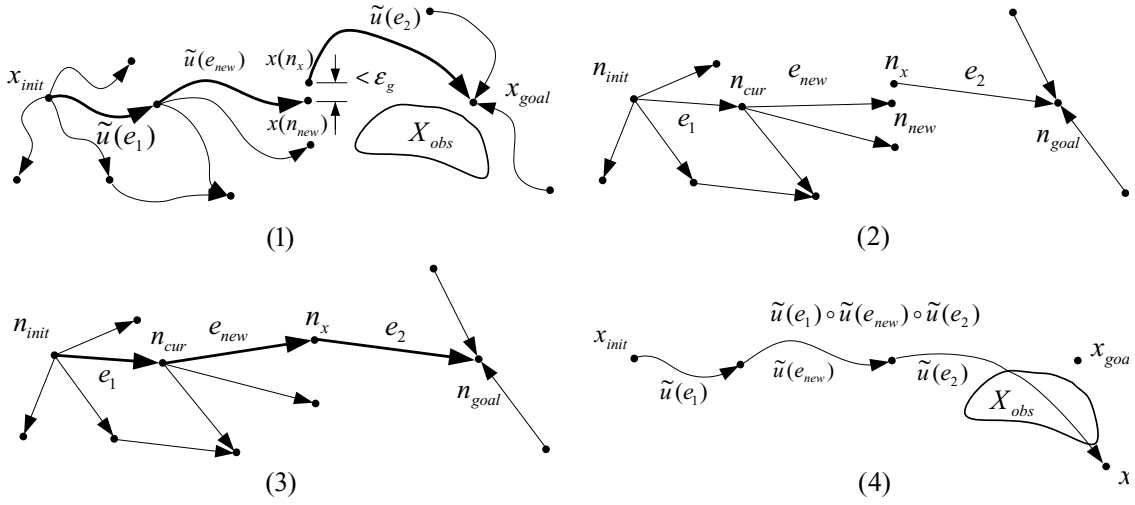


Figure 4.2: The generation and effects of a state mismatch

trajectory might be found for some positive η as shown in Fig. 4.3, in which the left picture shows the trajectories of controls that are shown in the right picture, the solid curves respectively represent a solution control and its trajectory, and dashed lines respectively represent the control that is constructed from sampled controls and its constructed trajectory. The constructed trajectory is only in the neighborhood of the solution trajectory. With inappropriate control sampling, these neighboring controls and trajectories might not be violation-free or have undesired clearance and tolerance.

The main idea of resolution completeness for sampling-based MPD could be explained through Figure 4.4, in which τ is a trajectory of a 5-stage solution with clearance w in the reachability graph, control mismatches cause only an approximate solution could be constructed as a solution path, and state mismatches further cause trajectory $\hat{\tau}_\xi$ of the solution path in the search graph that encodes the approximate solution to be discontinuous. Generally, to achieve resolution completeness, a planning algorithm must appropriately sample the continuous spaces and control these mismatches such that if trajectory τ of a K-stage solution with clearance w exists in the reachability graph, then a solution path that encodes the solution or its approx-

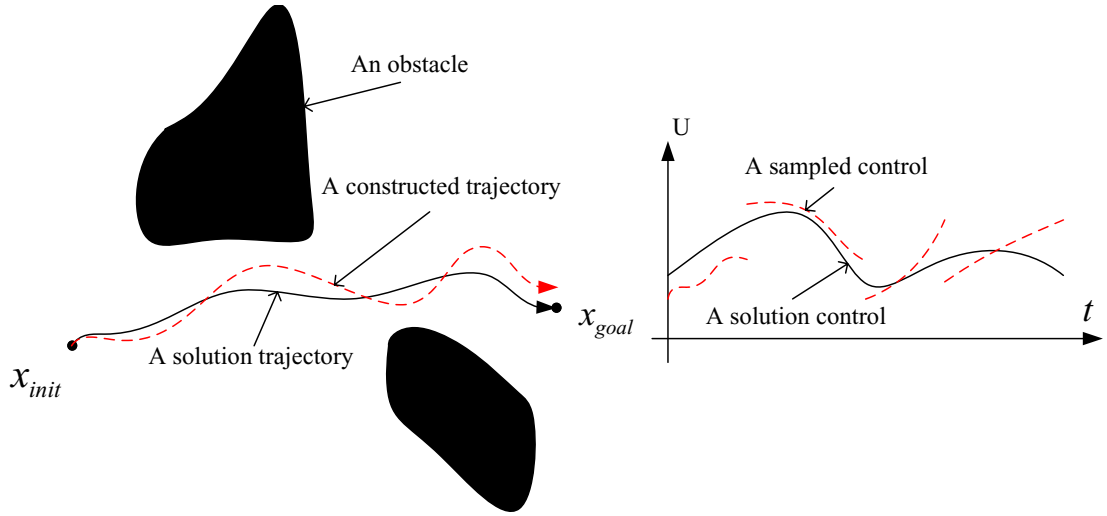


Figure 4.3: The effect of control mismatches

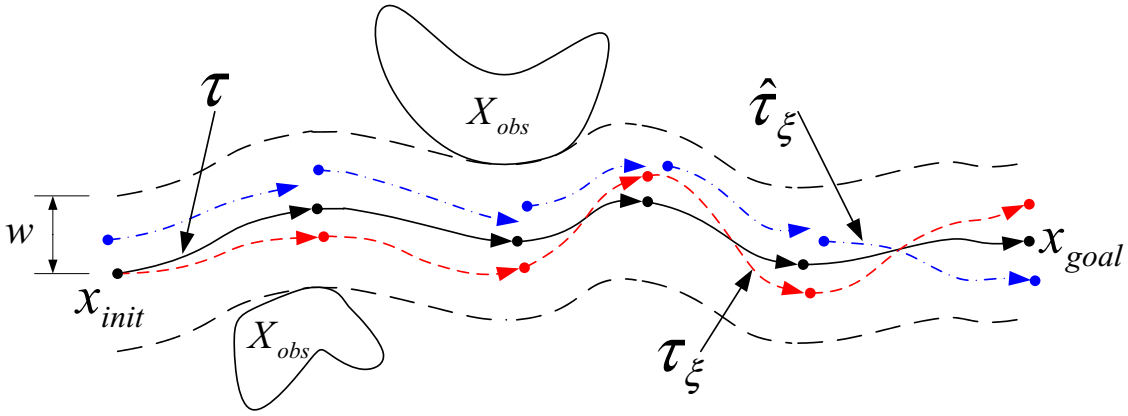


Figure 4.4: The main idea of resolution completeness for sampling-based MPD

imation will be constructed in finite time in the search graph. A precise definition for resolution completeness will be given in Section 4.2.2 after several terms are introduced in the later sections. In this thesis, the construction of the solution path is conservatively achieved by ensuring the distance $d_\tau(\hat{\tau}_\xi, \tau)$ from $\hat{\tau}_\xi$, i.e., the trajectory of the solution path, to τ, i.e., the trajectory of the solution, is less than w. The quantitative sufficient conditions are provided by first characterizing the mismatches and then deriving their relationship with the distance $d_\tau(\hat{\tau}_\xi, \tau)$ using Lipschitz conditions. In Section 4.2, a formal definition of resolution completeness through the reachability graph and the search graph is given. Sufficient conditions and their derivations are given in Sections 4.3 and 4.4. Finally, resolution completeness conditions are applied on two existing algorithms in Section 4.5.

4.1 Related Results

Our results are related to many techniques and concepts from a large amount of existing research [22; 35; 50; 66–68]. In [66], convergence of value functions computed by dynamic programming for discrete-time systems with infinite control set and continuous state space was analyzed. It is shown that the value function could be approximated with arbitrarily precision at a finite set of sampled points when resolution of the sampled set goes to infinity. Its analysis is based on Lipschitz conditions, which is similar to the analysis in this thesis. But the analysis here uses these conditions to obtain completeness conditions in the context of motion planning problems with continuous time systems.

In [35], an asymptotic form of completeness was presented: when the search depth is "big enough", resolution of discretization is "high enough", and the sampling rate for the control is "high enough", the planner will find an existing solution in finite time. In contrast, our resolution completeness analysis is *quantitative* in that specific

conditions are given in terms of resolution size and Lipschitz constants.

In [22; 47; 69], tracking lemmas show that by appropriately discretizing time and control space a solution for a given problem will always be closely approximated by a piecewise-constant acceleration control for a fully-actuated system. One nice thing about tracking lemma is that the discretization resolution is independent of the number of stages in a solution. It could provides better complexity results than the analysis in this thesis. Similar techniques are also used in [43]. However, these methods could only be applied on problems with fully-actuated systems. For an n-dimensional fully-actuated system, there are n independent inputs such that the theorem in [22] could be proved by n simultaneous independent tracking games in one-dimensional spaces. In [42], the planner for robots with L_2 dynamics bounds was improved to give the first known polynomial approximation algorithm for the curvature-constrained shortest-path problem in three or more dimensions. However, the mismatch between the initial state and the state of the initial state node is not well controlled such that applying the returned control might not drive the system to the intended goal state. Also, the correcting lemma requires a lower bound on the duration of the trajectories, such that solution trajectories with duration less than the lower bound might not be corrected with the discretized control set.

In [53; 54], the structure of reachable states for a discrete-time system with discrete control set, called *quantized control systems*, is studied. Conditions for the reachable set to be either dense or a lattice structure were studied for linear systems and nonholonomic chained-form systems with different discrete control set. Recently, [68] shows that with enough controls to achieve a reachable set with the lattice structure, any trajectories could be tracked with a given tolerance. However, the results are limited to driftless nilpotent systems.

A planner using a connecting local planner, e.g. [37], is generally not complete because many connecting local planners, e.g. steering methods, do not consider ob-

stacles and a small variation in the initial and goal states for the local planner might greatly change the trajectory connecting them. Examples of such local planners include many optimal control based local planners for nonholonomic systems [26; 28]. In [50; 67], topology properties of steering methods were introduced to reduce completeness of nonholonomic planning to that of path planning. A complete planner using the sinusoidal steering method as the local planner was also designed for chained form systems. In [70], an inverse kinematics-based planner for underactuated systems is proved to be complete using topology properties. In this thesis, a Lipschitz condition as a sufficient condition for topology property on the local planner is presented and used to provide precise resolution completeness conditions for general sampling-based MPD planners using connecting local planners.

In [36; 71], resolution completeness conditions for RRT-based planners were developed based on Lipschitz conditions of the motion equation and analysis of the reachability graph and the search graph. It provided quantitative completeness conditions for RRT-based MPD algorithms, i.e., with a given control sampling rate, and a maximum search depth, setting discretization resolution in a special range will guarantee resolution completeness. This thesis incorporates some of these ideas, and generalizes the work substantially to consider all possible sources that lead to incompleteness, applied to virtually any sampling-based planning algorithm.

4.2 Completeness via the Reachability Graph and Search Graph

Resolution completeness analysis is based on characterizing and controlling mismatches due to imperfections in the search process. In this section, various mismatches are explained and characterized, and a formal definition of resolution completeness of sampling-based MPD is explained through the relationship between these

two graphs.

4.2.1 Sources and characterization of mismatches

The sources and characterization of mismatches will be given with respect to the algorithm description in Chapter 3. The objective of the characterization is to provide an upper bound on these mismatches. Note that these sources are not listed in the order of appearance in the algorithm to avoid redundancy in the description.

- **State mismatches from state space discretization in Step 4**

When state space discretization is used, a state mismatch could be induced if the target node of a new edge is an existing node in the search graph when state x_{new} shares the same discretization set with the state of the existing node as shown in the middle picture of Fig. 3.4. The size of the state mismatch is measured by the distance between state x_{new} and state x_{n_x}. The size of the state mismatch due to the state space discretization is bounded by $2\epsilon_d$ when a state space sampling with discretization is used and has dispersion bound ϵ_d. As shown in Fig. 4.5, the left picture shows a state mismatch due to an explicit state space discretization, the dots in the right picture are in a maximal sampled set, and the circle represents the largest empty ball in the maximal sampled set. The radius of the circle is the dispersion of the maximal sampled set with respect to the state space. The dispersion bound is an upper bound on dispersions of all possible maximal sampled set. It can be seen that the size of the state mismatch will be bounded by twice of the dispersion bound.

- **State mismatches from search graph adjustment in Step 4**

A state mismatch is induced either when the state of a new node in one subgraph is in the ϵ_g neighborhood of the state of another node in another subgraph and two subgraphs are connected by unifying two nodes, or when the state of the new node

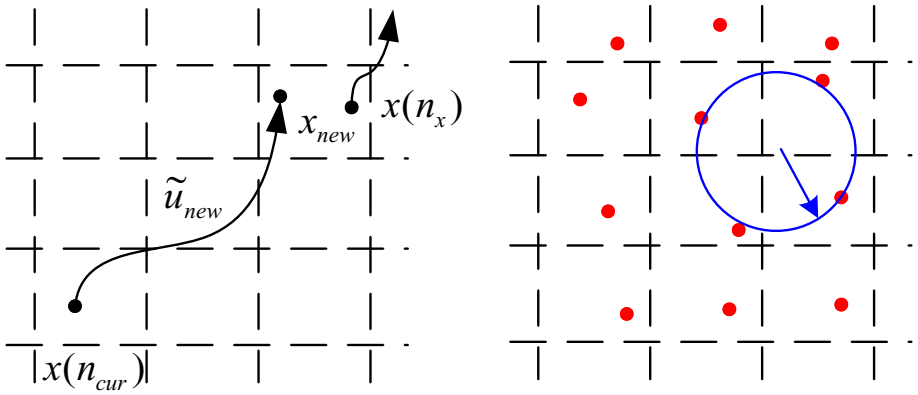


Figure 4.5: The relationship between the size of state mismatches and dispersion bound of state space sampling with explicit discretization

is in the ϵ_g neighborhood of a goal state. The size of these state mismatches is also bounded by $2\epsilon_g$.

- **State mismatches from search graph initialization in Step 1**

State mismatches will be induced if the states associated with the initial state node and goal state node are not the initial state and goal state, but just in their n_r neighborhoods. Similarly, the size of these mismatches is bounded by $2n_r$. Also, if the nodes in the initialized search graph are not associated with reachable states, state mismatches are also induced.

- **Control space sampling in Step 3**

In each iteration, a control is sampled from the sampling control set $\tilde{\mathcal{U}}$. These control mismatches could be parameterized based on whether a connecting local planner is used.

If an exact connecting local planner is used, the control space sampling is unified with the state space sampling under the assumptions that motion equations have a unique solution and the local planner returns a unique solution given two states. For each selected node n_{cur}, the sampling control set $\tilde{\mathcal{U}}$ normally includes an uncountably infinite number of controls, each of which connects state $x(n_{cur})$ to a state. However,

the sampled control set $\tilde{\mathcal{U}}_s$ could be at most countably infinite such that $\tilde{\mathcal{U}}_s$ is only a subset of $\tilde{\mathcal{U}}$ and control mismatches are induced. Since the control space sampling and state space sampling are unified, the control space sampling could use the characterization for the state space sampling. If a state space sampling with discretization is used and has dispersion bound ϵ_d, the size of the control space mismatch will also be bounded by $2\epsilon_d$.

If an approximate connecting local planner with tolerance ϵ_l is used, state mismatches are also induced besides control space mismatches when the final state of the trajectory of the sampled control from the state of the selected node is not the given state x_{new} as shown in the right picture of Fig. 3.2. The extra state mismatch will be characterized by ϵ_l.

If a non-connecting local planner is used, then the controls in the sampled control set are obtained by sampling in time and the input space described in Section 3.3.2. Therefore, these control mismatches are characterized by dispersion bound ϵ_t of time sampling and dispersion bound ϵ_u of the input space sampling.

- **Numerical calculations in Step 3 and 4**

To calculate the trajectory of a control from a starting state, integration of the control over the motion equation is extensively used in the MPD algorithms. Analytical integration is only available for few ODEs. When analytical integration is not possible, errors from numerical integration will cause state mismatches. The upper bound on these state mismatches is ϵ_i. If real numbers are approximated with floating point numbers in the computer, then state mismatches will also be induced. The floating point approximation is an important factor affecting completeness [72]. When floating point numbers are used, it is too complicated to explicitly manipulate floating point calculation error; therefore, it is assumed that these errors happen only when the state is stored into the search graph and a real positive number ϵ_n is the bound on the floating point calculation errors.

The state mismatches from state space discretization, search graph adjustment, and search graph initialization are called *discontinuities*. To simplify the derivation, the dispersion bound ϵ_d equals ϵ_r and ϵ_g such that the size of discontinuities will be bounded by $2\epsilon_d$. The state mismatches from numerical calculations and the approximate connecting local planners are called *state errors*.

4.2.2 Definition through the relationship between $\mathcal{G}$ and G

Because the sampling control set $\tilde{\mathcal{U}}$ is usually uncountable and the sampled control set $\tilde{\mathcal{U}}_s$ is countable, control mismatches exist, which implies that many possible controls are never attempted, or some nodes in the reachability graph $\mathcal{G}$ are not in the search graph G. Therefore, G converges to a subgraph of $\mathcal{G}$; in other words, G will match exactly some of nodes, edges and the associated states, trajectories of edges of $\mathcal{G}$ if control mismatches, discontinuities (except those induced when the state of the initialized nodes are not reachable states), or both exist. For the general setting, G will only converge to an approximation of a subgraph of $\mathcal{G}$ when discontinuities due to search graph initialization, or state errors exist.

A sampling-based MPD algorithm is resolution complete if for an MPD problem $\mathcal{P}$ and an approximation tolerance

$$0 < \epsilon_p < 1, \tag{4.1}$$

there exists some setting for parameters ϵ_d, ϵ_l, ϵ_i, ϵ_n, ϵ_u, and ϵ_t such that if a K-stage solution with clearance w and tolerance ϵ_s exists, a K-stage $(\epsilon_p w)$-approximation of the solution with clearance

$$(1 - \epsilon_p)w \tag{4.2}$$

and tolerance

$$\epsilon_s + w\epsilon_p \tag{4.3}$$

will be found in finite time. Usually, the precise range of parameter values for which completeness is guaranteed is not given (see the definition of resolution completeness in [18]). The theorems of Sections 4.3 provide these values for control space sampling and state space sampling, which is one of the main challenges that is addressed in the analysis.

Resolution completeness could be also defined in terms of the relationship between the reachability graph $\mathcal{G}$ and the search graph G. In Fig. 4.4, τ is the trajectory of a K-stage solution $\tilde{u}$ with clearance w and tolerance ϵ_s

$$\tilde{u} = \tilde{u}_1 \circ \tilde{u}_2 \circ \cdots \circ \tilde{u}_K, \tag{4.4}$$

in which $\tilde{u}_i$ is in the sampling control set $\tilde{\mathcal{U}}$ for $i = 1, 2, \cdots, K$. Control mismatches due to control space sampling might imply that only a trajectory τ_ξ of an η-neighboring solution from the initial state for some real positive η could be constructed with sampled controls. State mismatches might further cause τ_ξ to appear as a discontinuous trajectory $\hat{\tau}_\xi$ of a solution path that encodes the η-neighboring solution in G. Thus, *resolution completeness* of a sampling-based MPD planner means that for any

$$0 < \epsilon_p < 1, \tag{4.5}$$

if the trajectory τ of a K-stage solution with tolerance ϵ_s and clearance w exists in the reachability graph $\mathcal{G}$, then in finite time an $(\epsilon_p w)$-neighboring K-stage trajectory τ_ξ with clearance

$$(1 - \epsilon_p)w \tag{4.6}$$

and tolerance

$$\epsilon_s + w\epsilon_p \tag{4.7}$$

will be constructed as $\hat{\tau}_\xi$ of a solution path in the search graph G.

4.3 Sufficient Conditions

The main results are provided in two theorems in this section, which provide sufficient conditions on control space sampling and state space sampling to ensure resolution completeness for sampling-based planners either with or without using connecting local planners. It is assumed that only control mismatches and discontinuities exist. Conditions for other mismatches, such as state errors, are analyzed with similar techniques in Appendix A.4.

4.3.1 Assumptions for the theorems

Our results depend on combinations of some of the following assumptions. Assumptions 1, 2, and 3 are on the problem, and the others are on the algorithm.

Assumption 1 (Bounded slope controls) *Every coordinate of a control in the control space generator set $\bar{\mathcal{U}}$ is piecewise first-order differentiable, and the magnitude of its first derivative is bounded by a real nonnegative constant D_p.*

When a non-connecting local planner is used, the sampling control set is $\bar{\mathcal{U}}$. However, for general $\bar{\mathcal{U}}$ without a slope bound, it is impossible (shown in Lemma 19 in Appendix A.2) to sample a finite control set with arbitrarily small dispersion bounds ϵ_u and ϵ_t (as described in Section 4.2.1). Therefore, Assumption 1 is critical to ensure that a finite control space sampling in $\bar{\mathcal{U}}$ exists to achieve any given dispersion bounds ϵ_u and ϵ_t (this is shown in Lemma 22 in Appendix A). This assumption is reasonable in practice; controls normally have slope constraints because the input cannot change arbitrarily fast.

Assumption 2 (All controls in $\bar{\mathcal{U}}$ must cause nonzero state transitions)

$$d_{inf} = \inf_{x \in X, \tilde{u} \in \bar{\mathcal{U}}} \|x - \tilde{f}(x, \tilde{u})\| > 0. \tag{4.8}$$

This assumption enables that each stage of a solution could add an edge to the search graph when state space discretization is used. For some systems, this may require removing controls that do not cause a motion in a state space. For example, in a driftless system, a constant control that produces zero velocity is assumed to be removed.

If this assumption is not satisfied and state space discretization is used, the final state of the trajectory of a sampled control from the state of a selected node might share the same discretization set as the state of the selected node. The sampled control will not be associated with a new edge in the search graph because one discretization set could have at most one state in X_G.

Assumption 3 (Lipschitz condition on the motion equation) *The motion equation, f, in Eq.* (2.146) *satisfies the following Lipschitz condition with Lipschtz constant L_c:*

$$\|f(x,u) - f(x',u')\| \leq L_c(\|x - x'\| + \|u - u'\|) \tag{4.9}$$

for any x, x' in X and u, u' in U.

This assumption is the basis for bounding the distance between trajectories due to mismatches in Section 4.4.2. The motion equation of a large amount of robotic systems satisfies this condition. For detailed description of Lipschitz conditions, please refer to [52; 73].

For a simple example of a point robot that moves along a line, its motion equation is

$$\begin{cases} \dot{p} &= v \\ \dot{v} &= u \end{cases} \tag{4.10}$$

in which p is the position of the robot, v is its velocity, and u is the input. The state is

$$x = [p, v]^T, \tag{4.11}$$

and the state space is

$$X = \{[p, v]^T \mid p \in [0, 100], v \in [-5, 5]\} \tag{4.12}$$

The input space is

$$U = \{u \mid u \in [-2, 2]\}. \tag{4.13}$$

Written in vector form, motion equation becomes

$$\dot{x} = \frac{d}{dt}\begin{bmatrix} p \\ v \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} p \\ v \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u. \tag{4.14}$$

Using the infinity norm, the Lipschitz condition is

$$||f(x, u) - f(x', u')||_\infty \leq ||x - x'||_\infty + ||u - u'||_\infty, \tag{4.15}$$

for all $x, x' \in X$ and $u, u' \in U$ and Lipschitz constant, L_c, is 1.

An example of motion equation which does not satisfy the Lipschitz condition in Eq. 4.9 is as follows:

$$\begin{cases} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= \frac{1}{x_1} + u \end{cases}, \tag{4.16}$$

which is defined on state space

$$X = \{[x_1, x_2]^T | x_1 \in [-10, 10] \setminus \{0\}, x_2 \in [-10, 10]\} \tag{4.17}$$

and input space

$$U = \{u | u \in [-1, 1]\}. \tag{4.18}$$

It can be verified that for any given real positive L_c, the Lipschitz condition with Lipschtz constant L_c in Eq. (4.9) is not satisfied when x_1 is in the $(\frac{1}{\sqrt{L_c}})$-neighborhood of 0. However, if an open neighborhood of 0 of x_1 is removed from the state space of this system, a Lipschitz constant will exist.

Assumption 4 (Systematic search behavior) *A sampling-based MPD algorithm is systematic if at least one new node-edge pair, which corresponds to a state-control*

pair, will be explored in each iteration, every node-edge pair in the search graph will be explored only once, and the algorithm will stop when all node-edge pairs in the search graph are explored.

This assumption helps to ensure finite running time and represents a constraint on the way that the node selection and local planning parts of the algorithm behave together. One step toward achieving this assumption is to require the local planner not to use controls that have been tried previously on the same state. If none are available, then another node must be selected for expansion.

Only when systematic search is used, can these algorithms be resolution complete. Using randomized search, only a probabilistic completeness will be achieved [13]. However, randomness could be used to design the heuristic to guide the systematic search in a resolution complete planner [36].

Assumption 5 (Asymptotic finite sampling) *Both the state space sampling and control space sampling are finite sampling, i.e., only a finite reached set X_G and a finite sampled control set $\tilde{\mathcal{U}}_s$ will be generated. The dispersion bounds ϵ_d, ϵ_t, and ϵ_u of sampling in these two spaces could be less than any give real positive ϵ.*

It is a very important requirement on the planners to achieve resolution completeness in the sense that the finite sampling ensures the finite running time and the asymptoticness ensures that any solution can be approximated arbitrarily close by prescribing small enough dispersion. These sampling could be achieved by sampling with discretization as shown in Appendix A.3.2.

Assumption 6 (Complete behavior in updating the search graph) *For any node n_{cur} that is chosen for expansion, whenever a sampled control $\tilde{u}$ generates a violation-free trajectory from state $x(n_{cur})$ to x_{new}, then an edge e with control $\tilde{u}$ will always to be added to the search graph.*

This assumption is automatically satisfied if state space discretization is not used. However, when state space discretization is used, one way to achieve the complete behavior is given as follows. If an explicit discretization is used, the target node of e will be: 1) a new node n_{new} with state x_{new} if x_{new} is in a discretization set that contains no state of X_G; 2) some node $n \neq n_{cur}$ in G if x_{new} is in a discretization set that contains $x(n)$. If an implicit discretization is used, the target node will be: 1) a new node n_{new} if x_{new} is not in the given neighborhood of any states in X_G; 2) a node $n \neq n_{cur}$ in G if x_{new} is in the given neighborhood of $x(n)$.

With this assumption, a solution will not be lost because one stage of the solution is not associated with an edge in the search graph due to state pruning. To ensure this assumption is true, Assumption 2 is necessary such that when resolution of state space discretization is high enough, new state x_{new} will move out of the current discretization set and a new edge will be added.

Assumption 7 (Completeness and uniqueness conditions for exact connecting local planners) *Given two states: 1) if the planning algorithm relies on a connecting local planner, then the local planner must be complete* [1]*; in other words, in finite time the local planner will find a solution if one exists in the simplified problem, or report that no solution exists, and 2) if multiple solutions exist, then the same solution will be returned every time.*

The first condition helps to achieve the overall completeness of sampling-based planner using the connecting local planner, and the second condition ensures that the control space sampling could be unified with the state space sampling.

[1]Because the problem that local planner solves is simpler by relaxing some constraints in $\mathcal{P}$, such as ignoring geometry of the work environment, a complete local planner above usually does not solve $\mathcal{P}$.

Assumption 8 (Lipschitz condition on connecting local planners) *There exists some real positive ϵ_x and L_u such that for all x_s, x_e in X with*

$$\|x_s - x_e\| < \epsilon_x, \tag{4.19}$$

then

$$\sup_{t \in [0, \bar{t}(\tilde{u}'))]} \|\tau_{\tilde{u}'}(x_s, t) - x_s\| \leq L_u \|x_s - x_e\|, \tag{4.20}$$

in which $\tilde{u}'$ is a sampled control such that

$$x_e = \tilde{f}(x_s, \tilde{u}'), \tag{4.21}$$

and $\tau_{\tilde{u}'}(x_s, \cdot)$ is the trajectory of $\tilde{u}'$ from state x_s.

Assumption 8 provides a sufficient condition for topological property [50] of the local planner, and will be specially used to bound the distance between trajectories due to control mismatches for planners using connecting local planners.

4.3.2 Main results

The sufficient resolution completeness conditions are provided for planners with or without using connecting local planners, respectively.

Theorem 9 (Conditions for planners that use non-connecting local planners and exact calculation) *Suppose that an MPD problem $\mathcal{P}$ satisfies Assumptions 1, 2 and 3 and a sampling-based MPD planner satisfies Assumptions 4, 5 and 6. For any*

$$0 < \epsilon_p < 1, \tag{4.22}$$

if there is a K-stage solution with clearance w and solution tolerance ϵ_s, using state space sampling with discretization and dispersion bound ϵ_d, and control space sampling with dispersion bound ϵ_u, ϵ_t, a planner will find an ϵ_p-approximation of the solution

with clearance $(1-\epsilon_p)w$ *and tolerance* $\epsilon_s + \epsilon_p w$ *in finite time under the following conditions:*

$$\epsilon_u(L_d - 1) + \epsilon_t D_f + 4\epsilon_d < \frac{L_d - 1}{L_d^{K+1} - 1} w, \tag{4.23}$$

$$\epsilon_u(L_d - 1) + \epsilon_t D_f < \frac{L_d - 1}{L_d^{K} - 1} \epsilon_p w, \tag{4.24}$$

in which

$$D_f = \sup_{x \in X, u \in U} \|f(x, u)\|, \tag{4.25}$$

and

$$L_d = e^{L_c \epsilon_v}, \tag{4.26}$$

and

$$\epsilon_v = \sup_{\tilde{u} \in \tilde{\mathcal{U}}} \bar{t}(\tilde{u}). \tag{4.27}$$

Theorem 10 (Conditions for planners using exact connecting local planners and exact calculation) *Suppose that an MPD problem* $\mathcal{P}$ *is given and a sampling-based MPD planner satisfies Assumptions 4, 5, 7, and 8. For any*

$$0 < \epsilon_p < 1, \tag{4.28}$$

if there is a solution with clearance w *and solution tolerance* ϵ_s*, then using control space sampling and state space sampling with discretization and dispersion bound* ϵ_d*, a planner will find an* ϵ_p*-approximation of the solution with clearance* $(1-\epsilon_p)w$ *and tolerance* $\epsilon_s + \epsilon_p w$ *in finite time under the following condition:*

$$\epsilon_d < \min \left\{ \frac{\epsilon_p w}{1 + 2L_u}, \frac{\epsilon_x}{2} \right\}. \tag{4.29}$$

4.4 Proof of Main Results

In this section, Theorems 9 and 10 are proved. The proof has two parts. The first part shows existence of a solution path that encodes an existing solution or

its approximation. The second part shows the algorithm will terminate in finite time. The key to the first part is to associate the algorithm parameters with the distance from the trajectory of the solution path to the trajectory of the solution, which is obtained using Lipschitz conditions on systems and local planners. Before proving the theorems, the following lemmas are presented and will be used in the proof.

4.4.1 Conditions for finite running time

The following lemma shows sufficient conditions for a planner to terminate in finite time.

Lemma 11 *For a given problem $\mathcal{P}$ and dispersion bounds ϵ_d, ϵ_u, and ϵ_t, if a sampling-based planner satisfies Assumptions 4 and 5, then it will terminate in finite time.*

Proof: By Assumption 5, for any given dispersion bounds, the state space sampling and control space sampling will ensure that only a finite number of nodes exist in G and $\tilde{\mathcal{U}}_s$ is finite. Furthermore, Assumption 4 ensures that algorithm will terminate in finite time after every control in $\tilde{\mathcal{U}}_s$ is tried for the state of every node in G. □

4.4.2 Variation of trajectories due to mismatches

As illustrated in Fig. 4.4, mismatches should be controlled such that an η-neighboring trajectory τ_ξ could be constructed as $\hat{\tau}_\xi$ of a solution path in the search graph if a solution trajectory τ exists. For a trajectory τ of a solution with clearance w, if $\hat{\tau}_\xi$ is in w-tube of τ, τ_ξ could be associated with a constructed solution path. Therefore, the return of a neighboring solution depends on whether the distance $d_\tau(\hat{\tau}_\xi, \tau)$ from $\hat{\tau}_\xi$ to τ is less than clearance w.

To derive the relationship between $d_\tau(\hat{\tau}_\xi, \tau)$ and mismatches in the state space and control space, the following theorem [73] is used to derive the relationship between

the distance and mismatches. The theorem characterizes the amount of variation in a trajectory with respect to changes in the initial state and parameters of ODEs.

Theorem 12 *[73] Let $f(t,x)$ be piecewise-continuous in t and Lipschitz in x on*

$$[t_0, t_1] \times W \tag{4.30}$$

with a Lipschitz constant L, where $W \subset R^n$ is an open connected set. Let $y(t)$ and $z(t)$ be solutions of

$$\begin{cases} \dot{y} &= f(t,y) \\ y(t_0) &= y_0, \end{cases} \tag{4.31}$$

and

$$\begin{cases} \dot{z} &= f(t,z) + g(t,z) \\ z(t_0) &= z_0 \end{cases} \tag{4.32}$$

such that $y(t)$ and $z(t)$ are in W for all t in $[t_0, t_1]$. Suppose that

$$\|g(t,x)\| \leq \mu, \tag{4.33}$$

in which

$$(t,x) \in [t_0, t_1] \times W, \tag{4.34}$$

and μ is a real positive constant. This implies that for all t in $[t_0, t_1]$

$$\|y(t) - z(t)\| \leq \gamma e^{L(t-t_0)} + \frac{\mu}{L}(e^{L(t-t_0)} - 1), \tag{4.35}$$

in which γ is a real positive constant that satisfies

$$\|y_0 - z_0\| \leq \gamma. \tag{4.36}$$

The effect of a state mismatch over the trajectory of a sampled control With Theorem 12, the effect is described through the following lemma, which shows that the discrete motion equation, $\tilde{f}$, satisfies Lipschitz conditions on the state space, X.

Lemma 13 *For a given $\mathcal{P}$, if its motion equation, f, satisfies Lipschitz condition in Assumption 3, then for any y_0 and z_0 in X and $\tilde{u}$ in sampling control set $\tilde{\mathcal{U}}$ the discrete motion equation, $\tilde{f}$, (defined in Eq.* (2.150)*) satisfies*

$$\|\tilde{f}(y_0, \tilde{u}) - \tilde{f}(z_0, \tilde{u})\| \leq L_d \|y_0 - z_0\|, \tag{4.37}$$

in which L_d is defined in Eq. (4.26)*.*

Proof: Given a control $\tilde{u}$ in $\tilde{\mathcal{U}}$, the motion equation in Eq. (2.96) is changed into the following form

$$\dot{x} = f(x, \tilde{u}(t)) = f_{\tilde{u}}(t, x), t \in [0, \bar{t}(\tilde{u})] \tag{4.38}$$

Because $\tilde{u}$ is piecewise-continuous in t, and f is Lipschitz in x and u with a constant L_c, $f_{\tilde{u}}(t, x)$ is piecewise continuous in t and Lipschitz in x on

$$[0, \bar{t}(\tilde{u})] \times X \tag{4.39}$$

with a constant L_c. Let $y(t)$ and $z(t)$ respectively be solutions of Eq. (4.38) with starting states be y_0 and z_0. The functions $y(t)$ and $x(t)$ are also respectively the trajectories of $\tilde{u}$ from starting state y_0 and z_0. According to Theorem 12, for any $t \in [0, \bar{t}(\tilde{u})]$

$$\|y(t) - z(t)\| \leq \|y_0 - z_0\| e^{L_c t}, \tag{4.40}$$

by choosing γ in Eq. (4.35) to be $\|y_0 - z_0\|$.

From the definition of the discrete motion equation in Eq. (2.150), it can be seen that

$$\begin{aligned} \|\tilde{f}(y_0, \tilde{u}) - \tilde{f}(z_0, \tilde{u})\| &= \|y(\bar{t}(\tilde{u})) - z(\bar{t}(\tilde{u}))\| \\ &\leq \|y_0 - z_0\| e^{L_c \bar{t}(\tilde{u})} \\ &\leq \|y_0 - z_0\| e^{L_c \epsilon_v} \\ &= L_d \|y_0 - z_0\|, \end{aligned} \tag{4.41}$$

in which L_d is defined in Eq. (4.26). □

From Eq. (4.40) and the definition of the distance $d_\tau(y, z)$ between trajectory y and z in Eq. (2.110), the following inequality is derived

$$d_\tau(y, z) \leq \sup_{t=0}^{\bar{t}(\tilde{u})} \|y(t) - z(t)\| \leq L_d \|y_0 - z_0\|. \tag{4.42}$$

Therefore, the following corollary shows the relationship between the distance between two trajectories and a state mismatch at the starting state.

Corollary 14 *For a control $\tilde{u}$ in $\tilde{\mathcal{U}}$ and a discrete motion equation, $\tilde{f}$, satisfying Lipschitz condition in X with constant L_d (defined in Eq. (4.26)), if the starting state y_0 is changed to z_0 by a state mismatch, then*

$$d_\tau(y, z) \leq L_d \|y_0 - z_0\| \tag{4.43}$$

and

$$d_\tau(z, y) \leq L_d \|z_0 - y_0\|, \tag{4.44}$$

in which $y(\cdot)$ and $z(\cdot)$ are trajectories of $\tilde{u}$ from states y_0 and z_0, respectively.

The effect of a control mismatch over the trajectory of a sampled control For control mismatches, their effects will be considered differently according to whether a connecting local planner is used. If an exact connecting local planner is used, under Assumption 7, the sampling in the control space is done via that in the state space as described in Section 4.2.1. Furthermore, Assumption 8 shows that for a state pair with their distance less than ϵ_x, any state along the trajectory that connects the state pair will be in the $L_u \epsilon_x$ neighborhood of the starting state, which provides a bound on the distance between trajectories. For a non-connecting local planner, the effect of the control mismatch is shown in the following lemma.

Lemma 15 *Assuming that a given problem $\mathcal{P}$ satisfies Assumption 3, if*

$$\tilde{u} : [0, t_1] \to U \tag{4.45}$$

and

$$\tilde{u}' : [0, t_2] \to U \tag{4.46}$$

are any two controls in $\tilde{\mathcal{U}}$*, then for any* x_0 *in* X *the discrete motion equation* $\tilde{f}$ *satisfies:*

$$\|\tilde{f}(x_0, \tilde{u}) - \tilde{f}(x_0, \tilde{u}')\| \leq (L_d - 1)d_u + |t_1 - t_2|D_f, \tag{4.47}$$

in which L_d *is defined in Eq.* (4.26)*,* D_f *is defined in Eq.* (4.25)*,*

$$d_u = \rho(\tilde{u}, \tilde{u}'), \tag{4.48}$$

and $\rho(\cdot, \cdot)$ is defined in Eq. (2.88).

Proof: Without losing generality, t_1 is assumed to be no larger than t_2.

For a fixed starting state x_0 in X, two ODEs are obtained according to two controls $\tilde{u}$ and $\tilde{u}'$ as follows:

$$\begin{cases} \dot{x} = f(x, \tilde{u}(t)) = f_{\tilde{u}}(t, x), \ t \in [0, t_1] \\ x(0) = x_0; \end{cases} \tag{4.49}$$

$$\begin{cases} \dot{y} = f(y, \tilde{u}'(t)) = f_{\tilde{u}'}(t, y) = f_{\tilde{u}}(t, y) + g_{\tilde{u}, \tilde{u}'}(t, y), \ t \in [0, t_2] \\ y(0) = x_0. \end{cases} \tag{4.50}$$

Because f is Lipschitz in x and u with a constant L_c, for any

$$(t, x) \in [0, t_1] \times X, \tag{4.51}$$

$$\|g_{\tilde{u}, \tilde{u}'}(t, x)\| = \|f(x, \tilde{u}(t)) - f(x, \tilde{u}'(t))\| \leq L_c d_u. \tag{4.52}$$

Also, because $\tilde{u}$ and $\tilde{u}'$ are piecewise-continuous in t, $f_{\tilde{u}}(t, x)$ is piecewise-continuous in t and Lipschitz in x on $[0, \bar{t}(\tilde{u})] \times X$ with a constant L_c. Therefore, for any $t \in [0, t_1]$

$$\|y(t) - x(t)\| \leq d_u(e^{L_c t} - 1). \tag{4.53}$$

In the time interval $[t_1, t_2]$, the value of $\tilde{u}'$ is assumed to make $\|\dot{y}\|$ reach its supremum value D_f (because X is bounded and U are compact and f satisfies the

Lipschitz condition with X and U, D_f is finite) such that

$$\|y(t_2) - y(t_1)\| \leq D_f |t_2 - t_1|. \tag{4.54}$$

By the triangle inequality, the following inequality is obtained.

$$\begin{aligned} \|\tilde{f}(x_0, \tilde{u}) - \tilde{f}(x_0, \tilde{u}')\| &= \|x(t_1) - y(t_2)\| \\ &\leq \|x(t_1) - y(t_1)\| + \|y(t_2) - y(t_1)\| \\ &\leq d_u(e^{L_c t_1} - 1) + D_f |t_2 - t_1| \\ &\leq d_u(L_d - 1) + D_f |t_2 - t_1|, \end{aligned} \tag{4.55}$$

in which L_d is defined in Eq. (4.26). □

From the proof in Lemma 15, the right side of Eq. (4.47) provides an upper bound on the distance between two trajectories, which is formulated in the following corollary.

Corollary 16 *Assuming that a given problem $\mathcal{P}$ satisfies Assumption 3, if*

$$u : [0, t_1] \to U \tag{4.56}$$

and

$$u' : [0, t_2] \to U \tag{4.57}$$

are two controls in $\tilde{\mathcal{U}}$, then for any x_0 in X,

$$d_\tau(x, y) \leq (L_d - 1) d_u + |t_1 - t_2| D_f \tag{4.58}$$

and

$$d_\tau(y, x) \leq (L_d - 1) d_u + |t_1 - t_2| D_f, \tag{4.59}$$

in which $x(\cdot)$ and $y(\cdot)$ are trajectories of $\tilde{u}$ and $\tilde{u}'$ from state x_0, respectively.

Effects of a state mismatch over the trajectory of a k-stage control Now it is known how the mismatches affect the trajectory of a sampled control. The effects of a state mismatch over the trajectory of a k-stage control are manifested through the variation of the starting state and are shown in the following lemma.

Lemma 17 *For a k-stage control $\tilde{u}$ and a discrete motion equation, $\tilde{f}$, that satisfies Lipschitz condition in X with constant L_d (defined in Eq. (4.26)), if the starting state is changed from x_0 to x_0' with*

$$\|x_0 - x_0'\| \leq d_s \tag{4.60}$$

for some real positive d_s, then the distance from the trajectory τ' of $\tilde{u}$ from x_0' to the trajectory τ of $\tilde{u}$ from x_0 will be no larger than $L_d^k d_s$.

Proof: The result is obtained by induction on k using Corollary 14. □

4.4.3 Proof of Theorem 9

Assume that a K-stage solution $\tilde{u}_s$ with clearance w and tolerance ϵ_s is

$$\tilde{u}_s = \tilde{u}_1 \circ \tilde{u}_2 \circ \cdots \circ \tilde{u}_K, \tag{4.61}$$

in which $\tilde{u}_i$ is in the sampling control set $\tilde{\mathcal{U}}$. Its trajectory from the initial state is τ. The proof will show that for any

$$0 < \epsilon_p < 1, \tag{4.62}$$

a solution path with K edges

$$\{e_1(n_{init}, n_2), e_2(n_2, n_3), \cdots, e_K(n_K, n_{goal})\} \tag{4.63}$$

will be constructed in the search graph in finite time and its K-stage control

$$\tilde{u}_s' = \tilde{u}_1' \circ \tilde{u}_2' \circ \cdots \circ \tilde{u}_K' \tag{4.64}$$

with

$$\tilde{u}_i' = \tilde{u}(e_i) \tag{4.65}$$

is an $(\epsilon_p w)$-neighboring solution with clearance $(1 - \epsilon_p)w$ and tolerance $\epsilon_s + \epsilon_p w$.

The proof consists of two parts. The first part shows that the solution path will exist in the search graph. And the second part shows that the solution path will be constructed in finite time.

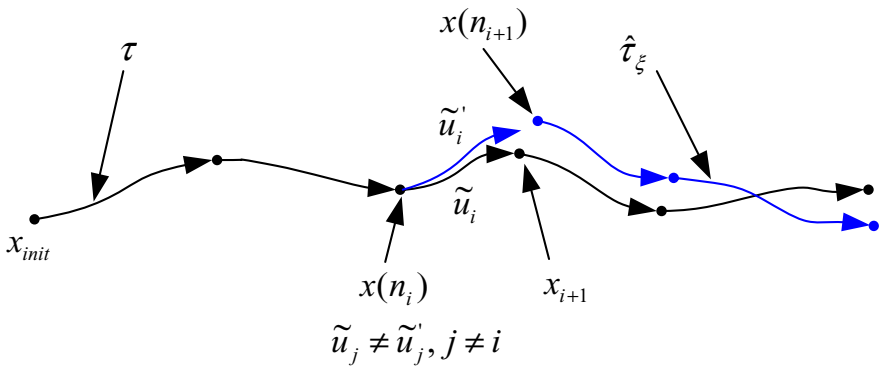


Figure 4.6: The trajectory $\hat{\tau}_\xi$ with one control mismatch and state mismatch

Existence of the solution path The trajectory of the solution path is

$$\hat{\tau}_\xi(\cdot) = \tau_{\tilde{u}_1'}(x(n_{init}), \cdot) \circ \tau_{\tilde{u}_2'}(x(n_2), \cdot) \circ \cdots \circ \tau_{\tilde{u}_K'}(x(n_K), \cdot). \tag{4.66}$$

To ensure that the solution path exists, $\hat{\tau}_\xi$ (shown in Fig. 4.4) should be violation-free, every $\tilde{u}_i'$ should be associated with an edge in the search graph, and the control of the solution path should satisfy the given tolerance and clearance.

Firstly, to make $\hat{\tau}_\xi$ violation-free, one sufficient condition is to require that trajectory $\hat{\tau}_\xi$ be in the w-tube of τ, i.e.,

$$d_\tau(\hat{\tau}_\xi, \tau) < w, \tag{4.67}$$

according to the definition of clearance in Section 2.3.3.

Assume that only one control $\tilde{u}_i$ does not equal $\tilde{u}_i'$ and the discontinuity only exists between the final state of the trajectory of $\tilde{u}_i'$ from state $x(n_i)$ and state $x(n_{i+1})$ for some i as shown in Fig. 4.6. By the given dispersion bounds for the control space sampling, there exists a sampled control $\tilde{u}_i'$ in $\tilde{\mathcal{U}}_s$ such that

$$|\bar{t}(\tilde{u}_i) - \bar{t}(\tilde{u}_i')| < \epsilon_t \tag{4.68}$$

and

$$\rho(\tilde{u}_i, \tilde{u}_i') < \epsilon_u, \tag{4.69}$$

and the discontinuity is bounded by $2\epsilon_d$. Since control $\tilde{u}'_j$ equals $\tilde{u}_j$ for $j \neq i$, the control

$$\tilde{u}_{+i} = \tilde{u}'_{i+1} \circ \cdots \circ \tilde{u}'_K = \tilde{u}_{i+1} \circ \cdots \circ \tilde{u}_K \tag{4.70}$$

$$\tilde{u}_{-i} = \tilde{u}'_1 \circ \cdots \circ \tilde{u}'_{i-1} = \tilde{u}_1 \circ \cdots \circ \tilde{u}_{i-1} \tag{4.71}$$

are $(K-i)$-stage and $(i-1)$-stage controls, respectively.

The bound on the distance from $\hat{\tau}_\xi$ to τ will be the maximum of the respective bounds of the distance from trajectories $\tau_{\tilde{u}_{-i}}(x_{init}, \cdot)$, $\tau_{\tilde{u}'_i}(x(n_i), \cdot)$, and $\tau_{\tilde{u}_{+i}}(x(n_{i+1}), \cdot)$ to trajectory τ since

$$\hat{\tau}_\xi(\cdot) = \tau_{\tilde{u}_{-i}}(x_{init}, \cdot) \circ \tau_{\tilde{u}'_i}(x(n_i), \cdot) \circ \tau_{\tilde{u}_{+i}}(x(n_{i+1}), \cdot). \tag{4.72}$$

Since there are no mismatches before stage i, the bound on the distance from $\tau_{\tilde{u}_{-i}}(x_{init}, \cdot)$ to τ is zero. Under Assumption 3, Lemma 15 will give

$$\begin{aligned} \|\tilde{f}(x(n_i), \tilde{u}_i) - \tilde{f}(x(n_i), \tilde{u}'_i)\| &\leq (L_d - 1)d_u + |\bar{t}(\tilde{u}_i) - \bar{t}(\tilde{u}'_i)|D_f \\ &< (L_d - 1)d_u + \epsilon_t D_f, \end{aligned} \tag{4.73}$$

and Corollary 16 bounds the distance between the trajectories of $\tilde{u}_i$ and $\tilde{u}'_i$ from state $x(n_i)$ as

$$d_\tau(\tau_{\tilde{u}'_i}(x(n_i), \cdot), \tau_{\tilde{u}_i}(x(n_i), \cdot)) \leq (L_d - 1)d_u + \epsilon_t D_f. \tag{4.74}$$

By triangle inequality, the distance between the final state x_{i+1} of the trajectory of $\tilde{u}_i$ from state $x(n_i)$ and state $x(n_{i+1})$ is bounded as follows

$$|x_{i+1} - x(n_{i+1})| \leq (L_d - 1)d_u + \epsilon_t D_f + 2\epsilon_d. \tag{4.75}$$

Furthermore, Lemma 17 gives the bound on the distance between the trajectories of $\tilde{u}_{K-i}$ from state x_{i+1} and $x(n_{i+1})$ as

$$d_\tau(\tau_{\tilde{u}_{K-i}}(x(n_{i+1}), \cdot), \tau_{\tilde{u}_{K-i}}(x_{i+1}, \cdot)) \leq (\epsilon_u(L_d - 1) + \epsilon_t D_f + 2\epsilon_d) L_d^{K-i}. \tag{4.76}$$

Therefore, the distance from $\hat{\tau}_\xi$ to τ is bounded as

$$d_\tau(\hat{\tau}_\xi, \tau) \leq (\epsilon_u(L_d - 1) + \epsilon_t D_f + 2\epsilon_d) L_d^{K-i}. \tag{4.77}$$

In the worst case, each control $\tilde{u}_i$ might not equal $\tilde{u}'_i$ and discontinuities could exist between the initial state and the state of the initial state node, the goal state and the state of the goal state node, and each edge of the solution path. Thus, by the triangle inequality, the distance from $\hat{\tau}_\xi$ to τ is bounded as

$$d_\tau(\hat{\tau}_\xi, \tau) \leq (\epsilon_u(L_d - 1) + \epsilon_t D_f + 4\epsilon_d)\frac{L_d^{K+1} - 1}{L_d - 1} \tag{4.78}$$

Choosing ϵ_d, ϵ_u, and ϵ_t to make the right side of Eq. (4.78) be less than clearance w will ensure that $\hat{\tau}_\xi$ is violation-free.

Secondly, with Assumptions 2 and 6, each $\tilde{u}'_i$ is associated with an edge in the search graph since $\hat{\tau}_\xi$ is violation-free such that the path could exist in G.

Lastly, to ensure that the control of the path has the required tolerance and clearance, the distance from trajectory τ_ξ of $\tilde{u}'_s$ from state x_{init} to τ must be bounded by $\epsilon_p w$. Because only control mismatches will affect $d_\tau(\tau_\xi, \tau)$, according to Lemmas 13, 15, 17, and the triangle inequality, the distance is bounded as

$$d_\tau(\tau_\xi, \tau) \leq (\epsilon_u(L_d - 1) + \epsilon_t D_f)\frac{L_d^K - 1}{L_d - 1}. \tag{4.79}$$

Choosing ϵ_d, ϵ_u, and ϵ_t to make the right side of Eq. (4.79) be less than $\epsilon_p w$ will ensure that the returned control satisfies the requirement.

Finite running time Under Assumptions 4 and 5, Lemma 11 shows that the algorithm will terminate in finite time with the dispersion bounds satisfying the above requirements since there are only a finite reached set X_G and sampled control set $\tilde{\mathcal{U}}_s$.

From the above description, given an approximation tolerance ϵ_p, if there is a K-stage solution with clearance w and tolerance ϵ_s, an ϵ_p-approximation of the solution will be constructed in the search graph in finite time, which completes the proof. □

4.4.4 Proof of Theorem 10

Assume that a solution

$$\tilde{u} : [0, t_f] \to U \tag{4.80}$$

with clearance w and tolerance ϵ_s exists. The proof is to show that for any

$$0 < \epsilon_p < 1, \tag{4.81}$$

a solution path could be constructed such that the control $\tilde{u}'$ of the path is an ϵ_p-approximation of $\tilde{u}$ and has clearance $(1 - \epsilon_p)w$ and tolerance $\epsilon_s + \epsilon_p w$.

Existence of the solution path With Assumptions 7 and 5, control space sampling is unified with state space sampling and characterized by the dispersion bound ϵ_d. The sampling could generate a finite set

$$S \subset X \tag{4.82}$$

with its dispersion no larger than ϵ_d. Therefore, for any t in $[0, t_f]$, $\tau(t)$ will be in the ϵ_d neighborhood of some state in A such that there is a finite state set

$$A = \{a_1 = x_{init}, a_2, \cdots, a_l\} \subset S \tag{4.83}$$

such that the image of τ is covered by the union of the ϵ_d neighborhood of each point in A. The elements in set A is indexed such that the nearest two neighbors of a_i are a_{i-1} and a_{i+1}. Thus, $\hat{\tau}_\xi$ could be constructed by connecting adjacent states in A as shown in Fig. 4.7. The dashed circles respectively represent the ϵ_d neighborhoods of points in set A. The trajectory of the solution from state x_{init} is τ. Points $\{b_j\}$ are along trajectory τ and respectively in the ϵ_d neighborhood of a_j in the order of $j = 1$, $2, \cdots$, and l. Specially, b_1 is chosen to be x_{init} and b_l is chosen to be $\tau(t_f)$. Therefore, trajectory τ is the concatenation of trajectories from b_j to b_{j+1} for $j = 1, 2, \cdots$, and $l - 1$. The trajectory $\hat{\tau}_\xi$ is constructed by approximating each trajectory from b_j to b_{j+1} by trajectory from a_j to a_{j+1}.

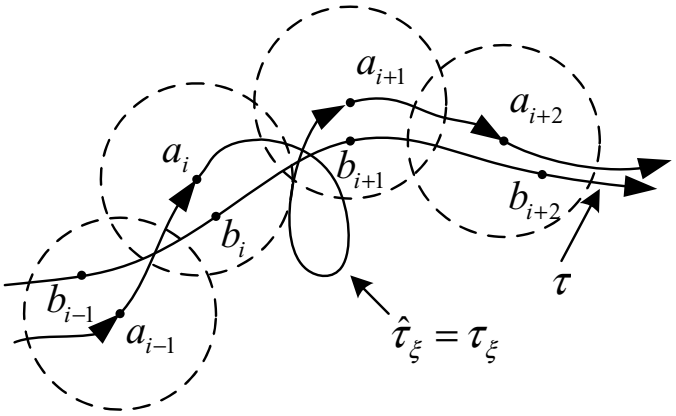


Figure 4.7: Construction of one segment of the trajectory of the solution path

In Fig. 4.7, an upper bound on the distance between a_i and a_{i+1} is

$$\|a_i - a_{i+1}\| \leq 2\epsilon_d, \tag{4.84}$$

since they are nearest neighbors and the neighborhoods of points in A cover the image of τ. By Assumption 8, choosing

$$2\epsilon_d < \epsilon_x \tag{4.85}$$

will make any point on the trajectory connecting a_i and a_{i+1} be in the $2L_u\epsilon_d$ neighborhood of a_i and in the $2L_u\epsilon_d + \epsilon_d$ neighborhood of b_i. Therefore, the distance from $\hat{\tau}_\xi$ to τ is bounded as

$$d_\tau(\hat{\tau}_\xi, \tau) < (2L_u + 1)\epsilon_d. \tag{4.86}$$

Because only control mismatches exist when the exact local planner is used, the distance from τ_ξ to τ equals that from $\hat{\tau}_\xi$ to τ, i.e.

$$d_\tau(\tau_\xi, \tau) = d_\tau(\hat{\tau}_\xi, \tau). \tag{4.87}$$

Choosing ϵ_d satisfying

$$(2L_u + 1)\epsilon_d < \epsilon_p w \tag{4.88}$$

will ensure the solution path exists and the control of the path has has clearance $(1 - \epsilon_p)w$.

With the state space sampling with dispersion bound ϵ_d, a_l in A will be in the ϵ_d neighborhood of $\tau(t_f)$ such that τ_ξ has tolerance $\epsilon_s + \epsilon_d$. Choosing ϵ_d satisfying Eq. (4.88) will also achieve the desired tolerance.

Finite running time With Assumptions 5 and 4, Lemma 11 shows that the algorithm will terminate in finite time with the given dispersion.

By combining the results from the above two steps, it can be observed that if there is a solution, a neighboring solution will be constructed as a solution path in finite time under the assumptions and conditions in the theorem. □

4.5 Applications to Particular Algorithms

This section applies resolution completeness conditions to improve the analysis of two planning algorithms in Section 3.2.1. The first planner [35] provided a qualitative form of resolution completeness. The second planner [13] is only probabilistically complete in its original form. It is hoped that these two examples illustrate how to apply our general analysis techniques to many other sampling-based planning algorithms.

4.5.1 Strengthening resolution completeness conditions for an existing planner

The algorithm in [35] has been presented in Section 3.2.1. The planner has a qualitative form of resolution completeness, i.e., the planner will find an existing solution in a finite number of iterations when setting δt small enough, search depth cutoff K large enough, and R large enough. However, no quantitative requirements on these parameters are provided. Theorem 9 is applied to obtain the requirement on R for a given δt, finite control set $\tilde{\mathcal{U}}_s$, and K.

Assumption verification Assume that the problem satisfies Assumption 2 and 3 and the control space generator set $\bar{\mathcal{U}}$ only consists of a finite number of constant controls with a fixed duration δt. The infinity norm $\|\cdot\|_\infty$ on X is used. The planner satisfies Assumption 4 since it is a systematic search. Assumption 5 can be verified using the techniques in Appendix A.3.2. However, the planner needs to be changed to satisfy Assumption 6. When state x_{new} is in an occupied parallelepiped which contains the state of node n_x in the search graph, instead of discarding x_{new} and adding no edge as shown in the left picture of Fig. 3.4, a new edge from node n_{cur} to node n_x is inserted as shown in the middle picture of Fig. 3.4.

Parameter setting Because $\bar{\mathcal{U}}$ is finite and $\tilde{\mathcal{U}}_s$ equals $\bar{\mathcal{U}}$, control mismatches do not exist, which implies that ϵ_u and ϵ_t are 0. By Theorem 9, choosing the dispersion bound of state space sampling with discretization

$$\epsilon_d = \frac{w(L_d - 1)}{4(L_d^K - 1)}, \tag{4.89}$$

in which

$$L_d = e^{L_c \delta t} \tag{4.90}$$

ensures that the algorithm will find a solution if one exists with tolerance ϵ_s and clearance w.

Using $\|\cdot\|_\infty$ norm, dispersion bound of state space sampling with discretization is no less than twice of the maximal parallelepiped width. Furthermore, the maximal parallelepiped width should be less than d_{inf} (defined in Eq. (4.8)) to ensure that Assumption 6 could be satisfied. Assume that the maximal distance along each dimension of the configuration is l_i; R should satisfy

$$R > \lceil \log_2 \max_{i \in \{1,\ldots,n\}} \frac{l_i}{c} \rceil, \tag{4.91}$$

in which

$$c = \min \left\{ \frac{\epsilon_d}{2}, d_{inf} \right\}, \tag{4.92}$$

in which $\lceil a \rceil$ denotes the *ceiling* that is the smallest integer no less than a.

4.5.2 Making a probabilistically complete planner become resolution complete

The algorithm in [13] is probabilistic completeness, i.e., the probability of finding an existing solution approaches one as the number of iterations approaches infinity.

Assumption verification Assume that the problem satisfies Assumption 2 and 3 and the control space generator set $\bar{\mathcal{U}}$ of $\mathcal{P}$ only consists of a finite number of constant controls with a fixed duration δt. The planner does not satisfy Assumption 4 because a node-edge pair could be repeatedly explored, one new node-edge might not be explored in each iteration, and the search is not exhaustive. Assumption 5 is violated since an infinite number of nodes could be generated by the state space sampling.

Four modifications to RRT-based planners are made [36] to satisfy these assumptions. The first three ensure that Assumption 4 is satisfied. The third one ensures that Assumption 5 is satisfied.

1. To avoid exploring the one node-edge pair more than once, exploration information is kept for every node in G. Specifically, assume that control $\tilde{u}$ extends state $x(n_{cur})$ of node n_{cur} in G to state x_{new}. If either x_{new} is associated with a new node or the trajectory to it violates constraints, then $\tilde{u}$ is marked as explored for node n_{cur} and will not be applied any more.

2. To explore one new node-edge pair in each iteration, if all the controls of a selected node n_{cur} is selected are marked as explored, then another node with unexplored controls in the search graph will be chosen and an unexplored control will be applied.

3. To ensure an exhaustive search, the algorithm will stop when either a solution is returned or a failure is reported if there is no nodes with unexplored controls.

4. To make the state space sampling be a finite sampling, an implicit discretization is used. Assume that x_{new} is the final state of a new generated violation-free trajectory of a sampled control $\tilde{u}$ from state $x(n_{cur})$ of node n_{cur}. Before associating state x_{new} with a new node n_{new} to G, the distance of x_{new} to the states of nodes in G with respect to the given norm $\|\cdot\|$ is calculated. For a given real positive constant α, if x_{new} is in the α neighborhood of state $x(n_x)$ of node $n_x \neq n_{cur}$ in G, then x_{new} is discarded, but an edge that connects node n_{cur} and n_x is added; otherwise, state x_{new} is associated with a new node n_{new} and the edge that connects n_{cur} and n_{new} is added.

Parameter setting Because $\bar{\mathcal{U}}$ is a finite set, ϵ_u and ϵ_t are 0. From Theorem 9, the dispersion bound should satisfy the following inequality

$$\epsilon_d < \frac{w(L_d - 1)}{2(L_d^K - 1)}. \tag{4.93}$$

Because the dispersion bound of the state space sampling with discretization is no less than the twice of α, and α should be less than d_{inf} to enforce Assumption 6, the algorithm is guaranteed to find a desired solution in finite time, if one exists with tolerance ϵ_s, by setting

$$\alpha < \min\left\{\frac{\epsilon_d}{2}, d_{inf}\right\}. \tag{4.94}$$

Chapter 5

Gap Problems and Planning with Gap Reduction

Even though sampling-based planning algorithms have solved many challenging problems [36–38], one common problem with these algorithms is that their solution paths could have discontinuities (also called gaps in this chapter) as described in Section 4.2.1. An example of the gap in bi-directional search is shown in Fig. 4.2. In bi-directional methods, the search graph initially consists of two disjointed subgraphs, which initially have the initial and goal state nodes, respectively. One expands forward-in-time, and the other one expands backward-in-time. If the distance between one new node n_{new} in one subgraph and another node n_x in the other subgraph (the distance between nodes is defined as that between two associated states) is less than a given gap tolerance, two subgraphs are connected by unifying these two nodes as one node. In the case shown in Fig. 4.2, the unified node is n_x. The solution path is from the initial state node to the goal state node and its gap is at edge e_{new}. Similarly, the search graph of PRM-based planners is initialized with multiple disjointed subgraphs. If states of two nodes from two subgraphs are close enough, two subgraphs are connected with a unified node. If a solution path exists and passes multiple subgraphs, it could have a gap for each unified node along the path.

Gaps greatly degrade the quality of the solutions, especially for bi-directional or PRM-based search. As shown in Picture (4) Fig. 4.2, when a small gap changes the staring state from $x(n_x)$ to $x(n_{new})$, the final state change greatly from x_{goal} to x_f after integration over the same control $\tilde{u}(e_2)$. Therefore, planners are expected to use small gap tolerances. However, under a finite control space sampling, systems will not be small-time locally controllable (STLC). In some cases, the sets of reachable states from the initial state have the structure of a lattice, which prevents the exact matching of the solution trajectory endpoint with the goal state. No solution path could be found if the smallest distance between the reachable states and the goal state is larger than the given gap tolerance. In cases in which the set of reachable states is everywhere dense [74], or even continuous [75], it is possible in principle to add a sequence of sampled controls to move the final state arbitrarily close to the goal state, but this is done at the expense of the efficiency of the trajectory and longer running time since there will be less solution paths and the search depth, i.e., the number of edges, of solution paths tends to increase.

Note that a planner could quickly return solution paths with a big gap tolerance because the solution paths have low search depth. If a separate algorithm could efficiently reduce the gaps, then planners can quickly find a solution by first finding solution path candidates with big gaps and then reducing their gaps. One way to reduce gaps is to use analytical solutions for steering problems, that is, to design a trajectory to connect two end states of the gap while ignoring obstacles. However, there are only few analytical solutions [26–28; 30; 32; 76]. Furthermore, the cost of the resulting trajectories might be dramatically increased at the gaps. For example, for the kinematically controllable system [30], the system needs to stop completely when switching between decoupling vector fields during the steering process.

Alternatively, a symmetry-based perturbation method is developed [59; 77] to reduce the gaps by perturbing the solution candidate. The comparison of gap reduction

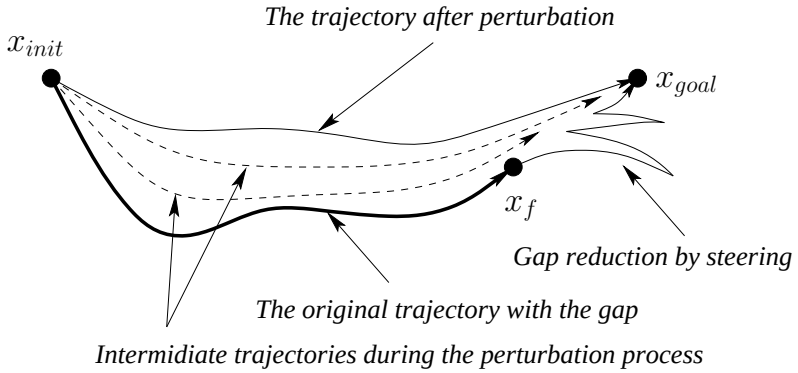


Figure 5.1: Comparison of gap reduction using perturbation and steering

by perturbation and steering is shown in Fig. 5.1. The gap reduction problem is cast as an optimization of the distance between two gap endpoints, which is efficiently evaluated by using symmetries of systems to avoid computationally expensive numerical integrations. One advantage of our method is that many local constraints on states and inputs could be naturally enforced in the trajectory after symmetry-based perturbations. Besides our work, a similar method is presented in [78] by tailoring the reactive path deformation method [79]. However, their methods did not use special geometric structures of the robotic systems such that their cost function evaluation could be very expensive and it is difficult to enforce the local constraints on states in the perturbed trajectories. In this chapter, the gap reduction algorithms is presented using symmetries, and combined with different types of sampling-based planners, which include the single and bi- directional RRT-based planner[36] and a PRM-based planner [14]. By comparing simulation results from different systems and problems, the new planners improved by gap reduction algorithms well outperform the original ones.

5.1 Gap Problems in Sampling-Based Algorithms

The generation of gaps has been described in Section 4.2.1. Gaps could seriously degrade the quality of a solution, especially for bi-directional search and PRM-based search. Therefore, it is expected to use small ϵ_g in sampling-based algorithms. However, when the gap tolerance ϵ_g decreases, the time to find a solution normally increases dramatically. For example, a single directional planner uses breadth-first search as node selection. If the state of a new node goes into the ϵ_g neighborhood of x_{goal}, a solution is returned. As ϵ_g decreases, the number of reachable states from x_{init} in the ϵ_g neighborhood of x_{goal} will decrease and the search depth of these reachable states will be higher, which means a longer running time to return a solution. In the worst case, if the reachable states with a set of sampled controls form a lattice structure and the smallest distance between x_{goal} and all reachable states is larger than ϵ_g, then the sampling-based algorithm cannot return a solution even though a solution does exist form the given problem $\mathcal{P}$.

5.2 Motion Planning with Gap Reduction

Our approach for the gap problems is to efficiently eliminate big gaps in solution path candidates by perturbation. Noticing that the sampled controls in the path candidates are only a small subset of the original control space $\mathcal{U}$, these sampled controls are perturbed in $\mathcal{U}$ to eliminate the gaps. With the gap reduction algorithms, paths with states of their final nodes outside the ϵ_g neighborhood of the goal state could also be transformed into solution paths. Therefore, a solution could be returned faster since the number of solution paths is increased and these paths have lower search depth. In the following section, the gap reduction algorithms are first described and then incorporated in sampling-based algorithms.

5.2.1 Gap reduction by perturbation

To be concise, a solution path candidate is considered, which only has one gap between the state of its final node and the goal state. Other types of gaps could be eliminated with minor modifications.

Assume that control of the path candidate is $\tilde{u}$, the objective of the gap reduction algorithm is to perturb $\tilde{u}$ into $\tilde{u}'$ such that the distance between the goal state and the final state x_f of the solution trajectory of $\tilde{u}'$ is less than a given tolerance and the new perturbed trajectory satisfies all constraints from the MPD problem. In this thesis, the gap reduction problem is cast as an optimization problem to minimize the distance between the final state and goal state by perturbing the parameters, which fully determine the final state.

For a control $\tilde{u}$, a given MPD problem, and a distance function, the outline of general gap reduction algorithms is given as follows.

1. **Parameterize Final State Calculation:** Determine a set of parameters, which determine the final state of the solution trajectory of the control under perturbation.

2. **Select a Subspace for Optimization:** Select a subset of parameters for the optimization. When the number of parameters for the final state is more than the number of parameters necessary to eliminate the gap, a subset of parameters could be chosen to avoid the expensive evaluation of high-dimensional gradient vectors in the optimization.

3. **Optimize in the Subspace:** Perturb the selected parameters to minimize the distance between the final state and the goal state without considering obstacles.

4. **Check Constraints:** Check whether the trajectory after the perturbation satisfies all constraints from the MPD problem. If not, discard the current perturbation.

5. **Check Gap Tolerance:** Check whether the gap distance is less than a given tolerance. If yes, report the solution; otherwise, go to Step 2 until a given number iterations is satisfied.

Because computation of the final state is extensively used in both evaluation and finite-difference gradient calculation of the gap distance in the optimization, its computation cost directly affects the effectiveness of the gap reduction algorithms.

In pure numerical gap reduction algorithms, the final state is calculated by integrating the perturbed control from the initial state. Since analytical integration is only available for few ODEs, expensive numerical integration is normally used to calculate the new final stats. The computation time for final states in each iteration grows linearly with the number of integration steps along a trajectory. Furthermore, with the above characterization, it is difficult to enforce the constraints on the states during the optimization process.

Therefore, instead of using the pure numerical gap reduction, we describe in the following sections about employing symmetries of the robotic systems to efficiently evaluate the final state such that the computation time for the final state is reduced to a constant-time (with respect to integration accuracy) operation. It will also be shown that many constraints on states and inputs are naturally maintained.

5.2.2 A class of systems with symmetries on principle fiber bundles

Symmetry is a fundamental geometric property of many robotic systems. Identification, properties and theorems of symmetry for general systems are out of the scope of this thesis. Refer to [51] for details. Only a general description of symmetry of robotic systems will be given, and then the scope of the method is limited to a class of systems with symmetries on principle fiber bundles [80], and finally use the car system in Eq. (2.159) to show as an example of a system in this class.

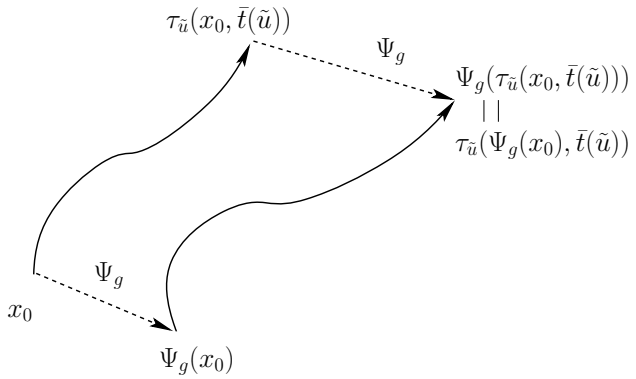


Figure 5.2: Group actions commute with state transitions

Consider a Lie group $\mathcal{G}$, with identity element e, acting on the state of the system through the (left) action

$$\Psi : \mathcal{G} \times X \to X; \tag{5.1}$$

the following shorthand will be often used

$$\Psi_g(x) := \Psi(g, x). \tag{5.2}$$

The group $\mathcal{G}$ is called a symmetry group for the system in Eq. (2.96) if the system's dynamics are *invariant* with respect to the action of $\mathcal{G}$. Invariance is equivalent to the following statement. Given any trajectory $\tau_{\tilde{u}}(x_i, \cdot)$ of control $\tilde{u}$ from state x_i which is a solution to Eq. (2.96), the trajectory $\Psi_g \circ \tau_{\tilde{u}}(x_i, \cdot)$ is also a solution to Eq. (2.96) for all $g \in \mathcal{G}$. It can be verified that invariance implies that group actions commute with state transitions. As shown in Fig. 5.2, if

$$x_f = \Phi_{\tilde{u}}(x_{init}, \bar{t}(\tilde{u})), \tag{5.3}$$

then

$$\Psi_g(x_f) = \Phi_{\tilde{u}}(\Psi_g(x_{init}), \bar{t}(\tilde{u})), \tag{5.4}$$

i.e.,

$$\Psi_g \circ \Phi_{\tilde{u}}(\cdot, \bar{t}(\tilde{u})) = \Phi_{\tilde{u}}(\cdot, \bar{t}(\tilde{u})) \circ \Psi_g. \tag{5.5}$$

In this thesis, a class of systems with symmetries on principle fiber bundles [80] are considered. State space X can be partitioned, at least locally, into the Cartesian product of two manifolds $X = \mathcal{G} \times \mathcal{Z}$, in which $\mathcal{Z}$ is the base space and $\mathcal{G}$ is the fiber space. For simplicity of exposition, it is assumed that $\mathcal{Z} = \mathbb{R}^{n_z}$, $n_z < n$. Accordingly, the generic point $x \in X$ is written as the pair $(g, z) \in \mathcal{G} \times \mathcal{Z}$. It is also required that the dynamics of these systems could be expressed in the following form:

$$\begin{cases} \dot{g} &= g\xi(z) \\ \dot{z} &= f_z(z, u) \end{cases}, \tag{5.6}$$

in which $\xi(z)$ is a representation of base variables as an element of $\mathfrak{g}$, the Lie algebra of $\mathcal{G}$. Such systems include many vehicles and moving robots, such as cars, submarines, and helicopters.

The car system with dynamics in Eq. (2.159) belongs to this class of systems and has symmetry group $SE(2)$. An element g in $SE(2)$ is a 3×3 matrix in the following form

$$g = \begin{bmatrix} \cos\theta & -\sin\theta & x \\ \sin\theta & \cos\theta & y \\ 0 & 0 & 1 \end{bmatrix}, \tag{5.7}$$

which denotes either a θ-angle rotation and $[x, y]^T$-translation, or position $[x, y]^T$ and orientation θ of a rigid body in the plane. State space X is locally partitioned into the Cartesian product of $SE(2) \times \mathbb{R}^2$, in which $SE(2)$ is the fiber space and $\mathbb{R}^2$ is the base space. State

$$x = [x, y, \theta, v_y, \omega]^T \tag{5.8}$$

is written into form (g, z), in which $g \in SE(2)$ represents (x, y, θ), and $z \in \mathbb{R}^2$ represents (v_y, ω). A group action $h \in SE(2)$ on state (g, z) is defined as

$$\Psi_h(g, z) = (hg, z). \tag{5.9}$$

It can be verified that the motion equation in Eq. (2.159) can be written in the

form of Eq. (5.6) with

$$\xi(z) = \begin{bmatrix} 0 & -\omega & v_x \\ \omega & 0 & v_y \\ 0 & 0 & 0 \end{bmatrix} \in \mathfrak{se}(2), \tag{5.10}$$

in which $\mathfrak{se}(2)$ is the Lie algebra of $SE(2)$.

5.2.3 Coasting trajectories of the class of systems

For the systems in Eq. (5.6), a state $x = (g, z)$ is called a *coasting state* if there exists an input u, called a *coasting input*, such that

$$f_z(z, u) = 0. \tag{5.11}$$

A trajectory $\tau_{\tilde{u}}(x_i, \cdot)$ of control $\tilde{u}$ from a coasting state $x = (g, z)$ is called a *coasting trajectory* if $\tilde{u}$ satisfies

$$\begin{cases} \tilde{u}(t) &= u \\ f_z(z, u) &= 0 \end{cases}, \tag{5.12}$$

for any t in $[0, \bar{t}(\tilde{u})]$. One advantage of coasting trajectories is that the base variables and input are kept constant along the trajectory. If the local constraints on the base variables and input are satisfied at the starting point, then they will also be satisfied along the trajectory if these constraints are independent of the fiber variables. Another advantage is that the fiber variables along the trajectory have the following explicit formulation:

$$g(t) = g(0) \exp(\xi(z)t). \tag{5.13}$$

The coasting trajectory from coasting state $x = (g, z)$ with coasting input u and the constant control $\tilde{u}$ are

$$\begin{cases} \tau_{\tilde{u}}(x, t) &= (g \exp(\xi(z)t), z) \\ \tilde{u}(t) &= u \end{cases}, \tag{5.14}$$

for any t in $[0, \bar{t}(\tilde{u})]$. The final state of the coasting trajectory differs from its starting state by a group action

$$\tau_{\tilde{u}}(x, \bar{t}(\tilde{u})) = \Psi_h(x), \tag{5.15}$$

in which

$$h = g \exp(\xi(z)t) g^{-1}. \tag{5.16}$$

For the car system in Eq. (2.159), the coasting state

$$x = [x, y, \theta, v_y, \omega]^T \tag{5.17}$$

and coasting input u satisfy

$$\begin{cases} 0 &= aC_r(v_y - b\omega) + av_x^2 M\omega + bC_r(v_y - b\omega) \\ \\ u &= \frac{C_r(v_y - b\omega) + v_x^2 M\omega}{C_f v_x} + \frac{v_y + a\omega}{v_x} \in U. \end{cases} \tag{5.18}$$

Furthermore, since the constraints on u, v_y and ω are independent of fiber variables x, y and θ, the constraints are also satisfied along the coasting trajectory. There is an analytical solution for the final state of the coasting trajectory for the car since $\exp(\xi(z)t) \in SE(2)$ can be calculated analytically.

5.2.4 Efficient gap reduction with symmetry

For the class of systems considered in the thesis, symmetry provides a decoupling between the base and fiber variables, i.e., the modification of the fiber variables of the starting state of a trajectory will not change the base variables of the trajectory. The decoupling provides us a way to eliminate the gap in the state space through two steps, which reduces the complexity of the gap reduction problem. In the first step, the gap in the base space is eliminated while ignoring the change of fiber variables. In the second step, the gap in the fiber space is eliminated. By the decoupling property, the base variables of the final state will be invariant in the second step, and therefore

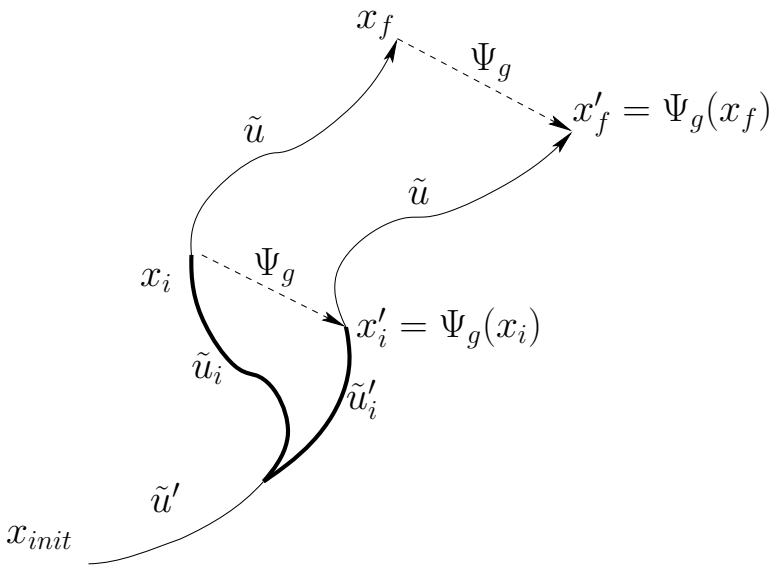


Figure 5.3: Perturbation of $\tilde{u}_i$ with symmetry to avoid reintegration of $\tilde{u}$ in calculating x'_f

the gap in the state space is eliminated at the end of the second step. Since the gap reduction in the base space is system specific, two examples are given along with two systems in Section 5.3.1. In the following, the gap reduction in the fiber space will be provided as a general technique for different systems with symmetries.

Efficient final state evaluation with symmetry The idea of the efficient final state evaluation is shown in Fig. 5.3, in which $\tilde{u}_i$ is perturbed into $\tilde{u}'_i$, x_i differs from x'_i by a group action Ψ_g, and the new final state x'_f is obtained by apply Ψ_g on the old final state x_f without reintegrating $\tilde{u}$ from x'_i. In this thesis, such perturbation is achieved by inserting coasting trajectories described in Section 5.2.3 into the trajectory after coasting states.

Assume that the solution trajectory of $\tilde{u}$ in Fig. 5.4 has three coasting states x_1, x_2, x_3 associated coasting inputs u_1, u_2, and u_3, respectively. These three states divide $\tilde{u}$ into four controls $\tilde{u}_1$, $\tilde{u}_2$, $\tilde{u}_3$, and $\tilde{u}_4$, which satisfies

$$\tilde{u} = \tilde{u}_1 \circ \tilde{u}_2 \circ \tilde{u}_3 \circ \tilde{u}_4 \tag{5.19}$$

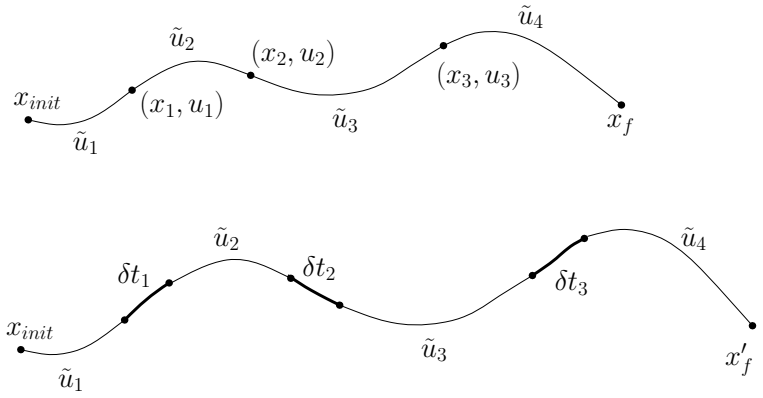


Figure 5.4: Perturbation by inserting coasting trajectories

and whose durations are t_1, t_2, t_3, and t_4, respectively. Let us define

$$\tau_{\tilde{u}}^{\bar{t}(\tilde{u})} = \tau_{\tilde{u}}(\cdot, \bar{t}(\tilde{u})) : X \to X. \tag{5.20}$$

Since the motion equation is time-invariant, the following is obtained

$$x_f = \tau_{\tilde{u}_4}^{t_4} \circ \tau_{\tilde{u}_3}^{t_3} \circ \tau_{\tilde{u}_2}^{t_2} \circ \tau_{\tilde{u}_1}^{t_1}(x_{init}). \tag{5.21}$$

Coasting trajectories of constant controls are inserted from these coasting states. These constant controls, denoted as $\tilde{u}'_1, \tilde{u}'_2$, and $\tilde{u}'_3$, have value u_1, u_2, and u_3 and durations δt_1, δt_2, and δt_3, respectively. The new final state is

$$\begin{aligned} x'_f &= \tau_{\tilde{u}_4}^{t_4} \circ \tau_{\tilde{u}'_3}^{\delta t_3} \circ \tau_{\tilde{u}_3}^{t_3} \circ \tau_{\tilde{u}'_2}^{\delta t_2} \circ \tau_{\tilde{u}_2}^{t_2} \circ \tau_{\tilde{u}'_1}^{\delta t_1} \circ \tau_{\tilde{u}_1}^{t_1}(x_{init}) \\ &= \tau_{\tilde{u}_4}^{t_4} \circ \Psi_{h_3} \circ \tau_{\tilde{u}_3}^{t_3} \circ \Psi_{h_2} \circ \tau_{\tilde{u}_2}^{t_2} \circ \Psi_{h_1} \circ \tau_{\tilde{u}_1}^{t_1}(x_{init}) \\ &= \Psi_{h_3} \circ \Psi_{h_2} \circ \Psi_{h_1} \circ \tau_{\tilde{u}_4}^{t_4} \circ \tau_{\tilde{u}_3}^{t_3} \circ \tau_{\tilde{u}_2}^{t_2} \circ \tau_{\tilde{u}_1}^{t_1}(x_{init}) \\ &= \Psi_{h_3} \circ \Psi_{h_2} \circ \Psi_{h_1}(x_f), \end{aligned} \tag{5.22}$$

in which

$$h_i = g_i \exp(\xi(z_i \delta t_i)) g_i^{-1}, \tag{5.23}$$

and

$$x_i = (g_i, z_i), i = 1, 2, 3. \tag{5.24}$$

In Eq. (5.22), the second equality comes from Eq. (5.15), that is, the final state of the coasting trajectory differs from its starting state by a group action. The third equality

comes from Eq. (5.5), that is, that group actions commute with state transitions. Note that 1) the calculation of the new final state only needs to evaluate the group actions of coasting trajectories without reintegrating the original controls. The group actions needs at most the numerical integration of the coasting trajectories, and could even be evaluated analytically for some cases, such as when the symmetry group is $SE(3)$. 2) The new final state is fully parameterized by nonnegative durations of coasting trajectories.

Calculating the durations of the coasting trajectories to achieve the desired group action is called *trajectory planning via inverse kinematics* in [30], which generally has numerical solutions except for few cases [81].

5.2.5 Selection of a subspace for optimization

Generally, if a gap exists in a space of dimension n_g, perturbing n_g parameters of the final state could eliminate the gap. If the number of parameters for the final state is larger than n_g, it is important to determine which n_g parameters could better eliminate the gap.

Observing that most gradient-based optimization techniques employ the steepest descent method as their starting step, it is expected that a nonlinear program with better convergence rate at the starting point for the steepest descent method might have a good chance to converge faster using other optimization techniques. Therefore, the convergence rate $\frac{1}{\alpha}$ of the steepest descent method for the nonlinear program at the starting point is calculated [82] as follows:

$$\alpha^2 = 1 - \frac{(\sum_i \varsigma_i^2 \lambda_i^2)^2}{(\sum_i \varsigma_i^2 \lambda_i^3)(\sum_i \varsigma_i^2 \lambda_i)}, \tag{5.25}$$

in which $\{\lambda_i\}$ are eigenvalues of Jacobian matrix J of the final state with respect to perturbation parameters and $\{\varsigma_i\}$ are coefficients of linear decomposition of $x_f - x_{goal}$ with respect to eigenvectors of J. Multiple subsets of parameters are chosen and their

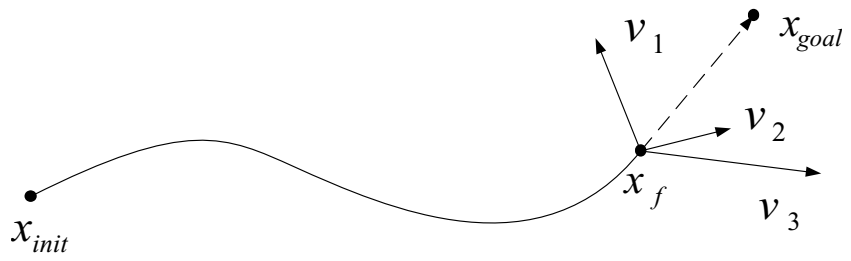


Figure 5.5: The intuition of subspace selection with Eq. (5.25) for optimization

convergence rates are evaluated. The subset with the highest convergence rate will be used for optimization.

The intuition of Eq. (5.25) is shown in Fig. 5.5. Assume that the final state has three parameters and the gap is in a 2-dimensional space. Vectors v_1, v_2, and v_3 are the partial derivatives of the final state with respect to three parameters, respectively. Apparently, perturbation using parameters of v_2 and v_3 would less likely to move the final state to the goal state than using parameters of v_1 and v_3, that is, the convergence rate of v_2 and v_3 will be less than that of v_1 and v_3.

5.2.6 Incorporating gap reduction algorithms with planners

To show the effect of gap reduction algorithms, they are combined with three types of sampling-based planning algorithms, which are respectively single and bi- directional RRT-based planning algorithms [36], and PRM-based planning algorithms [14].

The basic single- and bi-directional RRT-based algorithms have been described in Sections 3.2.1 and 3.2.2. From the algorithm description for RRT-based planners, it can be seen that the gaps are only induced in checking whether the distance between two states is less than the gap tolerance. Given two state x_1 and x_2, and assume that state x_1 is associated with node n_f in the tree containing the initial state node, the modification to check whether there is a solution path through these two states is as follows:

1. **Find Solution Path Candidate:** Check whether the distance between two

states is less than a given big gap tolerance. If not, report no solution path passes through these two states; otherwise, we have a path candidate.

2. **Eliminate Base Space Gap:** Retrieve the control $\tilde{u}$ of the path from the initial state node to node n_f. Check whether the gap in the base space between state x_2 and the final state x_1 of the solution trajectory of $\tilde{u}$ can be eliminated. If not, report no solution path passes through these two states; otherwise, construct $\tilde{u}'$, the final state of its solution trajectory has no gap with x_2 in the base space.

3. **Eliminate Fiber Space Gap:** Use the perturbation method described in Section 5.2.1 to eliminate the gap between state x_2 and the final state of the solution trajectory of $\tilde{u}'$. If the gap distance is less than the given small gap tolerance, report that a solution path exists; otherwise, report no solution path passes through these two states.

PRM-based search algorithms have been successfully applied to solve many challenging motion planning problems without differential constraints. PRM-based planners have two phases. The first construction phase is to build a search graph to capture the connectivity of the collision-free configuration space. In the second query phase, the planners try to connect the initial and goal states to the search graph and find a path from the initial state to the goal state. An important part of the algorithm is to connect a state to its neighboring states. Without differential constraints, the connection of two states could be done by simply using straight lines. However, for MPD problems, each of connection corresponds to a challenging TBVP problem. Since analytical solutions are only available for few cases, sampling-based single-query algorithms, such as RRT-based planners, could be used to generate connection trajectories for two given states. However, the gaps in these connecting trajectories could greatly degrade the quality of the returned solution and using small gap tolerance dra-

matically increases the running time. Therefore, the above sampling-based planning algorithms with gap reduction can be used as the connection method for PRM-based planning method, which could be used for systems for which no steering algorithm exists.

5.3 Simulation Studies

To check the performance improvement of sampling-based planners with gap reduction algorithms, gap reduction algorithms were incorporated with single directional, bi-directional, and PRM-based sampling-based algorithms. These improved algorithms tried to solve different problems with systems. All the simulations were done on a 2GHz Pentium 4 PC running Linux Fedora Core 3.

5.3.1 Systems used in simulations

Two systems are considered in our simulation. One system is the trailer system, which is a nonholonomic system [32]. The other system is the car with dynamics in Eq. (2.159). The base error of the car system is eliminated analytically as follows. The car system has the following linear base dynamics.

$$\dot{z} = Az + Bu, \tag{5.26}$$

in which

$$A = \begin{bmatrix} -\frac{C_f+C_r}{v_x M} & \frac{C_r b - C_f a}{v_x M} - v_x \\ \\ \frac{C_r b - C_f a}{v_x I} & \frac{b^2 C_r + a^2 C_f}{v_x I} \end{bmatrix}, \tag{5.27}$$

and

$$B = \begin{bmatrix} -\frac{C_f}{M} \\ \\ \frac{C_f a}{I} \end{bmatrix}. \tag{5.28}$$

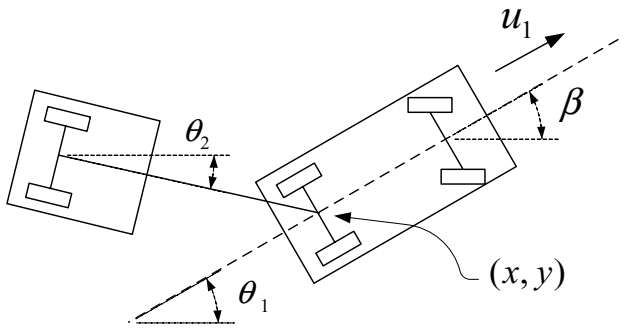


Figure 5.6: The sketch of the trailer system

Applying two constant controls with value c_1 and c_2 over two time intervals of the same duration δt from base variable z_1, the final base variable z_2 is obtained as follows:

$$z_2 = A_f z_1 + B_f u_d, \tag{5.29}$$

where

$$A_f = A_d^2, \tag{5.30}$$

$$B_f = [A_d B_d, B_d], \tag{5.31}$$

$$A_d = \exp(A\delta t), \tag{5.32}$$

$$B_d = \int_0^{\delta t} \exp(A(\delta t - t))B dt, \tag{5.33}$$

and

$$u_d = [c_1, c_2]^T. \tag{5.34}$$

Therefore, given two set of base variables z_1 and z_2, the value of two constant controls to eliminate the base space gap can be calculated.

In some cases, the calculated input is not in the input space. We could keep doubling δt and repeat the above calculation until an admissible input is available.

The trailer system shown in Fig. 5.6 consists of a car pulling a trailer. Its state is

$$[x, y, \theta_1, \beta, \theta_2]^T, \tag{5.35}$$

in which

$$x \in [0, 400], \tag{5.36}$$

$$y \in [0, 400], \tag{5.37}$$

and

$$\theta_1 \in [-\pi, \pi] \tag{5.38}$$

are x-, y-coordinates and orientation of the car,

$$\beta \in [-0.6, 0.6] \tag{5.39}$$

is the steering angle of the car, and

$$\theta_2 \in [-\pi, \pi] \tag{5.40}$$

is the orientation of the trailer. The orientation θ_1 and θ_2 must satisfy

$$|\theta_1 - \theta_2| < \frac{\pi}{2}. \tag{5.41}$$

The inputs to the system are u_1 and u_2, in which

$$u_1 \in [0, 2.0] \tag{5.42}$$

is the forward velocity, and

$$u_2 \in [-0.24, 0.24] \tag{5.43}$$

is velocity of the steering angle. Note, u_1 is intentionally set to be nonnegative so that the system is not STLC, and the gap cannot be reduced by trivially moving along the direction of Lie bracket. The motion equation of the trailer system are as follows:

$$\begin{cases} \dot{x} &= u_1 \cos(\theta_1) \\ \dot{y} &= u_1 \sin(\theta_1) \\ \dot{\theta}_1 &= \frac{u_1 \tan(\beta)}{L_1} \\ \dot{\beta} &= u_2 \\ \dot{\theta}_2 &= \frac{u_1 \sin(\theta_1 - \theta_2)}{L_2} \end{cases}, \tag{5.44}$$

in which

$$L_1 = 2.0 \tag{5.45}$$

is the length of the car,

$$L_2 = 10.0 \tag{5.46}$$

is the length of the hitch. By introducing the transformation

$$\theta_d = \theta_1 - \theta_2, \tag{5.47}$$

the last equation in (5.44) is changed to

$$\dot{\theta}_d = u_1 \tan(\beta)/L_1 - u_1 \sin(\theta_d)/L_2. \tag{5.48}$$

Then, the fiber variables are (x, y, θ_1), and the base variables are (β, θ_d). The coasting input and state satisfy that

$$u_2 = 0 \tag{5.49}$$

and

$$\tan(\beta)/L_1 - \sin(\theta_d)/L_2 = 0. \tag{5.50}$$

Given two set of base variables (β, θ_d) and (β', θ'_d) as the end points of the gap, it is assumed that

$$\theta'_d > \theta_d \tag{5.51}$$

without losing generality. The base space gap of the trailer system can be eliminated in three steps: 1) Set u_2 be the maximal value and u_1 be zero to increase β to its maximal value while keeping θ_d constant. 2) Set u_2 be zero and u_1 be maximal to increase θ_d to θ'_d while keeping β constant. 3) Set u_2 be the minimal value and u_1 be zero to decease β to β' while keeping θ_d constant.

5.3.2 Sensitivity of planner performance to the gap tolerance

To check the relationship between the performance of sampling-based planners without gap reduction and the gap tolerance, the bi-directional RRT-based planner [36] was used to solve a problem with the car with dynamics with gap tolerances at

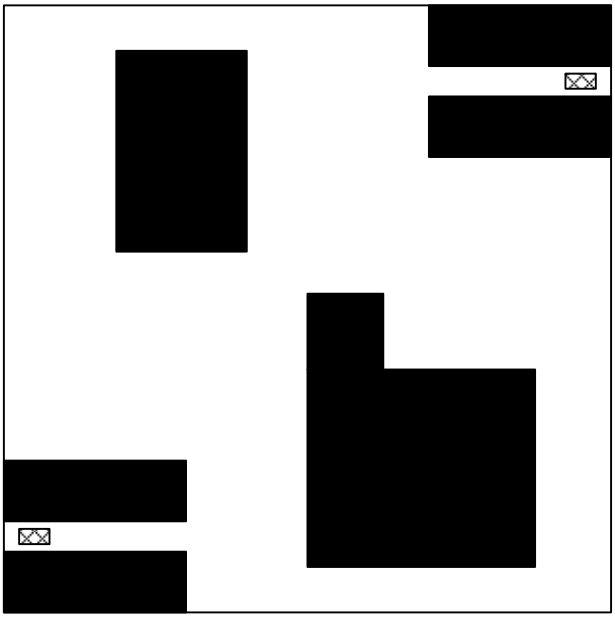


Figure 5.7: A problem with the car with dynamics for bi-directional planners

$100, 10, 1$, and 0.1, respectively. The problem is shown in Fig. 5.7, in which the initial and goal configurations are at the bottom-left and top-right corners, respectively, and the initial and goal velocities are zero. The distance between two gap endpoints in the simulations of this thesis is calculated using the following weighted Euclidean metric

$$\sum_{i=1}^{n}(x_i - x_i')^2 w_i \tag{5.52}$$

in which n is the dimension of the state space, $\{x_i\}$ and $\{x_i'\}$ are the state variables, and w_i is the weight for each dimension. The weights for different systems could be changed. For the 5-dimensional car, the weights are 1,1,1,1, and 100, which are respectively for v_y, ω, x, y, and θ in Eq. (2.159). The weight for θ is specially chosen to be 100 since a small variation in orientation could greatly change the final state of a trajectory.

For each tolerance, twenty trials were used to solve the problem. Each trial either returns a solution or report failure after 400000 iterations. The running time and the number of returned solutions are shown in Table 5.1. From the results, it can be seen

ϵ_g	Max.	Min.	Avg.	Suc.
100	158.7	2.6	29.7105	20
10	7113.7	2.6	1992.7	20
1	60736.2	2075.8	30576.3	11
0.1	61785.7	60168.1	60736.2	0

Table 5.1: The running time (in seconds) of planners with different gap tolerance, in which "Max.", "Min.", and "Avg." denote the maximal, minimal, and average time, and "Suc." denotes the number of returned solutions over 20 runs

that the performance of the sampling-based planners degrades dramatically with the gap tolerance decreasing.

5.3.3 Comparisons of gap reduction with or without symmetry

To compare the gap reduction algorithms with and without using symmetries, the pure numerical gap reduction algorithm was also implemented, in which the final states are evaluated by numerically integrating the perturbed controls. Both gap reduction algorithms were incorporated with single-directional RRT-based planners and test on problems in Figures 2.9 and 5.8. The initial and goal configurations for the problem in Fig. 5.8 are respectively above and below the black bar. For each problem, twenty trials were run. The overall running time and number of numerical integrations were reported. The gap tolerance is chosen to be 0.1 or $1e-6$. Each numerical integration is over a fixed time interval 0.01. The weights of the gap distance for the trailer system are 1, 1, 10, 1, and 10, which are respectively for x, y, θ_1, β, and θ_2 in Eq. (5.44).

Parameters for the final states of trajectories of controls from planners in [36] are as follows. Since the local planners in [36] use constant controls with fixed duration as sampled controls to generate new states. Therefore, each sampled control can be

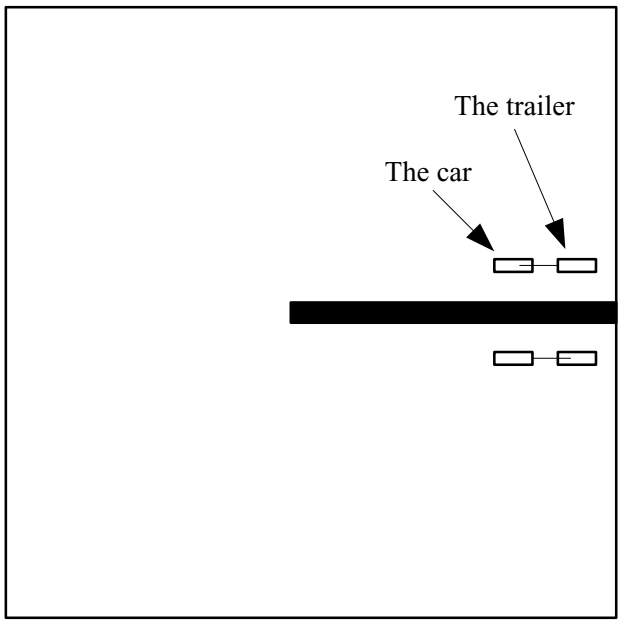


Figure 5.8: A problem with the trailer system for single-directional planners

characterized by $m + 1$ parameters, in which m of them denotes the value of the m-dimensional input and one is for the duration of the control. If a path candidate consists of k edges, then the control of the path will consist of $(m + 1)k$ parameters.

The simulation results are shown in Table 5.2. It can be seen that the planners using gap reduction with symmetry used less time and numerical integration. In the last row in the table, no overall running time and number of integration are obtained since the simulation is terminated when the planner using gap reduction without symmetry failed to return a solution in four days in one trial. In the analysis of the results, a large amount of trajectories returned from the gap reduction without symmetry failed to be a solution since the local constraints on states along the trajectories are violated, especially for the constraints in Eq. (5.41) for the trailer, which is the main reason that the planner using the gap reduction with symmetry has much better performance than that using the gap reduction without symmetry.

$\mathcal{P}$	Ob.	Sy.	ϵ_g	Time	Num.	Su.
Car	N	Y	0.1	753.0	8.2e7	20
Car	N	N	0.1	23091.3	4.5e9	20
Car	Y	Y	1.0e-6	36845.1	8.3e7	20
Car	Y	N	1.0e-6	94199.8	1.1e10	20
Tra.	N	Y	0.1	225.5	9.4e6	20
Tra.	N	N	0.1	243508.0	4.7e10	20
Tra.	Y	Y	1.0e-6	303.5	1.0e7	20
Tra.	Y	N	1.0e-6	-	-	-

Table 5.2: The running time (in seconds) and number of integration for planners improved by gap reduction with or without symmetry, in which "Ob." denotes whether obstacles are considered, "Tra." denotes the problem with the trailer system, "Sy." denotes whether symmetry is used, "Time" is the overall running time, "Num." is the overall number of integration, "Su." is the number of returned solutions over 20 runs

5.3.4 Effects of subspace selection for optimization

To see the effect, one gap reduction algorithm was implemented, for which the subspace is randomly generated. The planners using gap reduction with or without subspace selection were used to solve the problems with the car and the trailer system in Figures 2.9 and 5.8 for 20 runs. To concentrate the comparison of the effects of subspace selection, obstacles in the problems are ignored. The overall running time, the number of calls for the NAG optimization function, and the number of returned solutions are reported in Table 5.3. From the table, it can be seen that the overall time and number of calls in the planner using gap reduction with subspace selection is apparently less than those in the planner using gap reduction without subspace selection.

$\mathcal{P}$	Sel.	ϵ_g	Time	Num.	Suc.
Car	Y	1.0e-6	2395.3	1351	20
Car	N	1.0e-6	3678.0	5275	20
Tra.	Y	1.0e-6	457.0	3272	20
Tra.	N	1.0e-6	607.0	13304	20

Table 5.3: The effects of subspace selection, in which "Sel." denotes whether the subspace selection is used, "Time" is the overall running time in seconds, "Num." is the overall number of calls to the NAG optimization function, "Suc." is the number of returned solutions over 20 runs

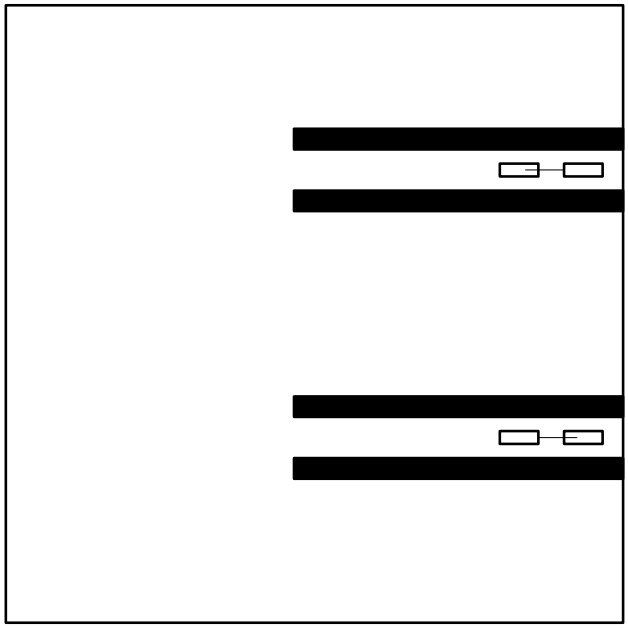


Figure 5.9: A problem with the trailer system for bi-directional planners

$\mathcal{P}$	ϵ_g	Time	Suc.
Car	1.0e-6	4294.2	20
Trailer	1.0e-6	15888.2	20

Table 5.4: The results of using the improved bi-directional planners to solve problems, in which "Time" denotes the overall running time, and "Suc." denotes the number of returned solution over 20 runs

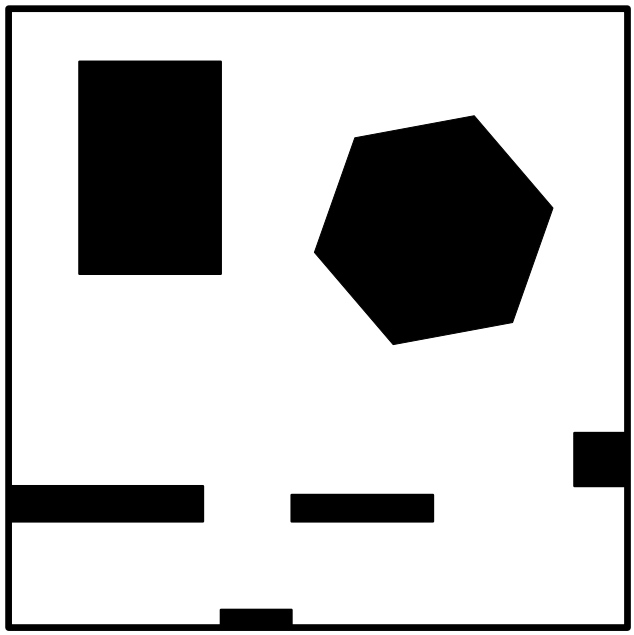


Figure 5.10: A problem with the car model for the PRM-based planner

5.3.5 Performance improvement of single- and bi-directional planners

The gap reduction algorithm was combined with the single directional RRT-based planner and test on problems in Figures 2.9 and 5.8. The gap tolerance is set to be $1.0e-6$. For each problem, the original planner and the improved planner using gap reduction with symmetry tried twenty runs. The original planner did not return any solution after 400000 iterations. The running time and number of returned solutions are shown in Table 5.2.

The gap reduction algorithm was combined with the bi-directional RRT-based planner and test on problems in Figures 5.7 and 5.9. The initial and goal configurations for the problem in Fig. 5.9 are at the top and bottom half of the picture. The gap tolerance is set to be $1.0e-6$. For each problem, the original planner and the improved planner using gap reduction with symmetry tried twenty runs. The original planner did not return any solution after 400000 iterations. The running time and number of returned solutions are shown in Table 5.4.

V.N.	C. T.(s)	Q. T.(s)	Suc.	E.N.
50	2243.3	45333.5	5	32
100	7847.6	57520.4	7	154
150	17387.5	32817.7	13	363
200	26507.0	88226.3	18	586
250	40331.9	138218.0	31	795

Table 5.5: Simulation results for the PRM-based planner, in which "V.N." means the number of vertices, "C.T." is the construction time, "Q.T." and "Suc." are respectively the overall query time and number of returned solutions for 40 queries, and "E.N." is the number of edges in the roadmap

5.3.6 A PRM-based MPD algorithm with gap reduction

The gap reduction algorithm was combined with the basic PRM-based planner and test on the problem in Fig. 5.10 with the car model in Eq. (2.159). The gap tolerance is chosen to be $1.0e-6$. The construction and query processes were alternatively run. Every time, after inserting the roadmap 50 new vertices, 40 queries were tried for randomly chosen initial and goal state pairs. To reduce the construction time, a sampling point tries to connect to at most 20 neighbors and each connection runs for 2000 iterations in the construction phase. In the query phase, a sampling point tries to connect to at most 40 neighbors and each connection runs for 20000 iterations to fully utilize the constructed roadmap. The results are shown in Table 5.5, for which it can be seen that the PRM-based method could be used to solve MPD problems even an analytical steering method is not available.

Chapter 6

Reducing Metric Sensitivity for RRT-Based Planners

RRT-based planners [61; 83] as sampling-based algorithms have been proposed recently for MPD problems. An RRT achieves rapid exploration by iteratively sampling a random state in the state space and extending the nearest state in the RRT toward the random state with a set of sampled controls. Various RRT-based planners have been designed recently for autonomous vehicles motion planning [84] and for nonlinear underactuated vehicles [85].

In spite of the successes of RRTs, one of the key shortcomings is the sensitivity of their performance with respect to a chosen metric. The metric helps to select a node in the search graph in each iteration, whose state is closest to the randomly generate state, i.e., it is a Voronoi bias heuristic. As shown in Fig. 6.1, the tree structures in four pictures are the RRTs and the partition of the plane is the Voronoi graph of the states in the reached set after generating the RRT for different number of iterations for a path planning problem. The nodes on the frontier of the RRT tend to have larger Voronoi regions such that they have higher probability to be chosen and explore more space. It can be seen that RRT achieves fast exploration in problems without differential constraints. However, for systems that involve differential con-

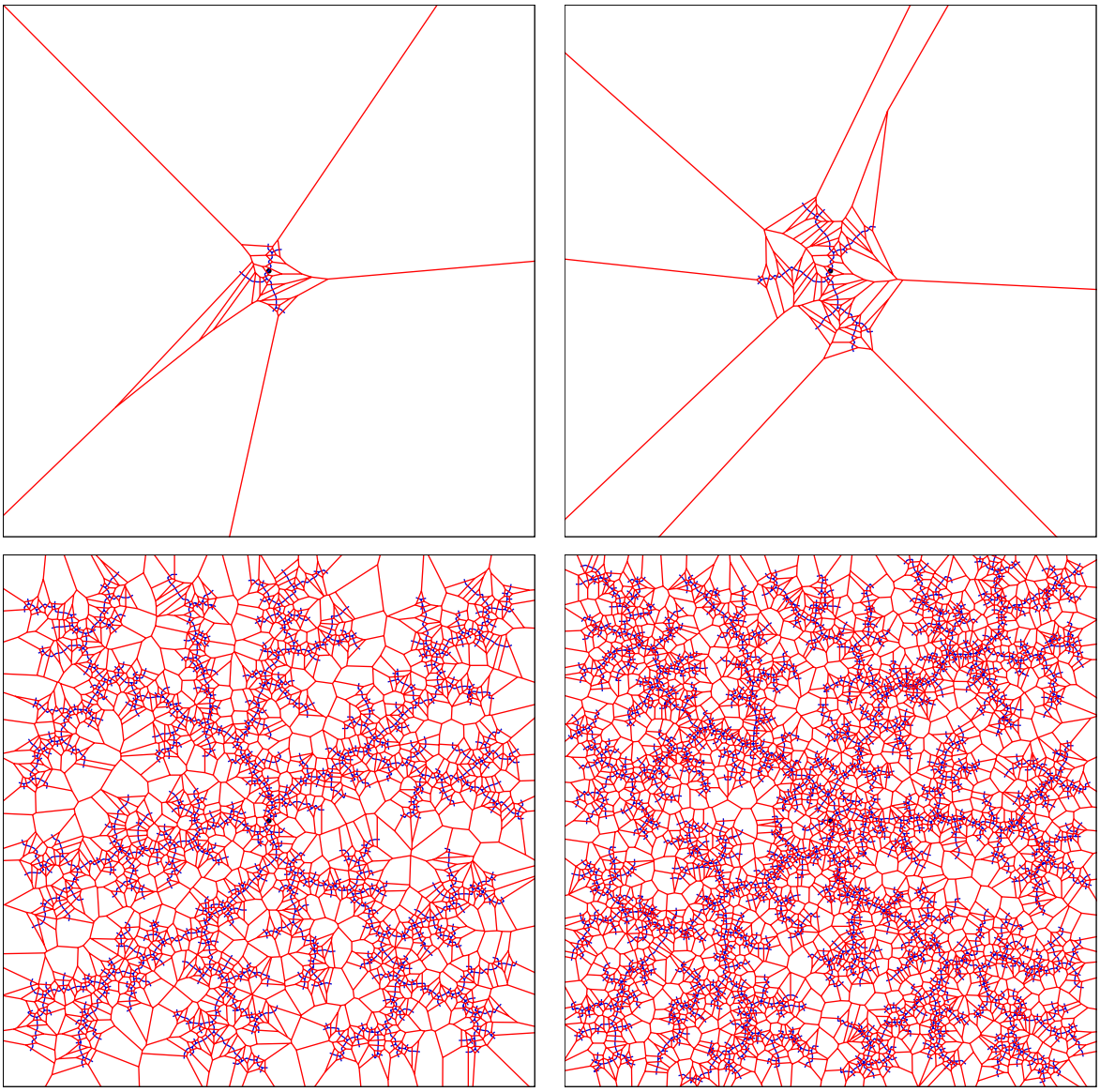


Figure 6.1: The fast exploration of RRT for a path planning problem

straints, the metric might 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 [84]), but in general there is a need to develop sampling-based MPD algorithms that achieve reliable performance in spite of a poor metric. It was this demand that leads to the emergence of the method presented here.

6.1 Metric Issues in RRT-Based Planners

The ideal metric for two states is the optimal cost-to-go, which is the optimal cost for the robot to move from one state to the other state. Calculating the optimal cost-to-go is at least the same difficulty as solving the MPD problems. Both differential constraints and global constraints have to be considered. The effect of differential constraints on the metric can be seen from the following example. Suppose a car-like robot is driving forward with high speed. The radius of the smallest circle in which it can turn is 100 meters. The robot is driving along the x axis and past the origin of y axis to the positive direction of the x axis. One state is at the 150 meters and another is at -100 meters on the x axis. A Euclidean metric might cause the planner to prefer the -100-meter state; however, it is the wrong decision given that the car cannot drive backward and would have to turn around to go to the -100-meter state; the state at 150 meters is closer in terms of the true cost. To understand the effect of global constraints on the metric, imagine that a robot is in a labyrinth of nonconvex obstacles. Two states might be close in terms of a Euclidean metric, but the correct metric should use the shortest path within the labyrinth.

The performance of RRT-based planners degrades because they select the node and sample controls solely depending on the given metric function. If differential constraints exist, then it is difficult for RRT-based planners to avoid obstacles in the

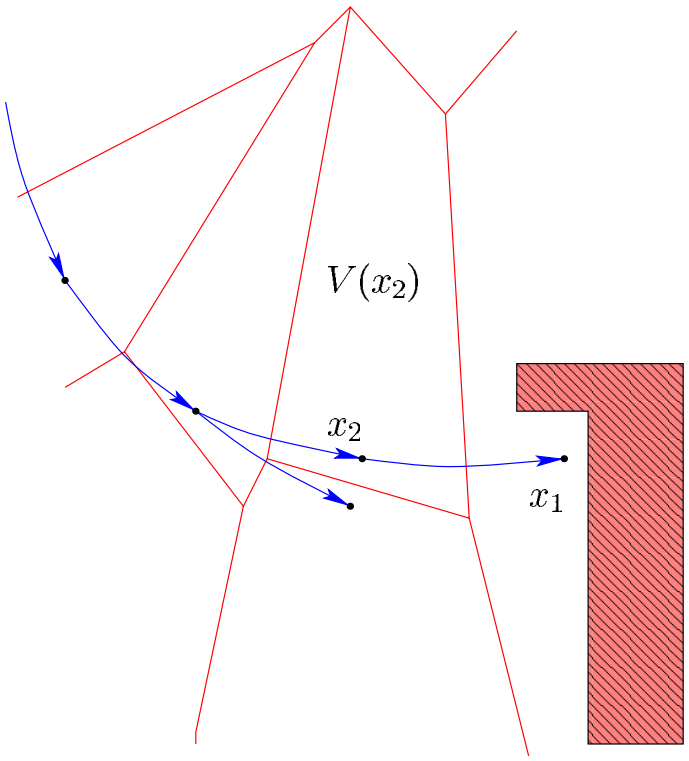


Figure 6.2: The issue in selecting the node in RRT-based planners

planning process. In node selection, RRT chooses node n_{near}, whose state $x(n_{near})$ is the closest state in X_G to a randomly generated state x_{random}. Under differential constraints, many nodes might be chosen numerous times, even though trajectories from their states will be in collision. However, it could be difficult to choose some nodes, from which a solution path could be obtained. As shown in Fig. 6.2, the curves represent the trajectories associated with edges of the search graph, the dots denote the states in the reached set X_G, and the straight line segments compose the Voronoi graph of the states in the reached set X_G using the given metric. The system will always be in collision if it is in state x_1 but could avoid the obstacle from state x_2. If the node selection solely depend on the given metric, then state x_2 is hardly to be chosen since state x_1 has a much larger Voronoi region that has a higher probability to contain the random state x_{random}.

The generation of trajectory segments of the RRT also only relies on the given

metric ρ, which further decreases the chance of avoiding obstacles in the exploration. Even though the state of a chosen node has the chance to avoid an obstacle, the control sampling might generate a control that drives the system along a trajectory in collision. The control that yields good exploration might be discarded because x_{new} derived from this control has "larger" distance to x_{rand} than that of the other states derived from the other controls. An example of this problem is shown in Fig. 6.3, in which x_1, x_{21}, and x_{22} are final states of trajectories of three sampled controls from state x_2, the system in state x_{22} will always run into a collision, and the dot lines compose the Voronoi graph of these three final states. Assume that RRT chooses node n_2 with state x_2 when the random state x_{random} is the Voronoi region $V(x_2)$ of state x_2. RRT will choose a sampled control, whose final states is the closest to the random state x_{random}. This control sampling strategy could also be interpreted with the Voronoi region of x_1, x_{21}, and x_{22}. A sampled control will be chosen if state x_{random} is in the Voronoi region of the state the control generates. To choose the control to generate state x_{21} in Fig. 6.3, state x_{random} has to be in the Voronoi region $V(x_{21})$ of state x_{21}. Therefore, to avoid the obstacle, the random state x_{random} has to be in the intersection of $V(x_2)$ and $V(x_{21})$, which could be an event with small probability.

6.2 Adaptive Reduction of Metric Sensitivity

This method in this thesis is not to design a system-specific metric, but to collect information during the exploration and expand $\mathcal{T}$ according to a simple metric ρ and information collected. It can make RRT less metric-sensitive and therefore more robust.

The following information is collected during the search:

Exploration information: For each RRT-node, whether a control has already

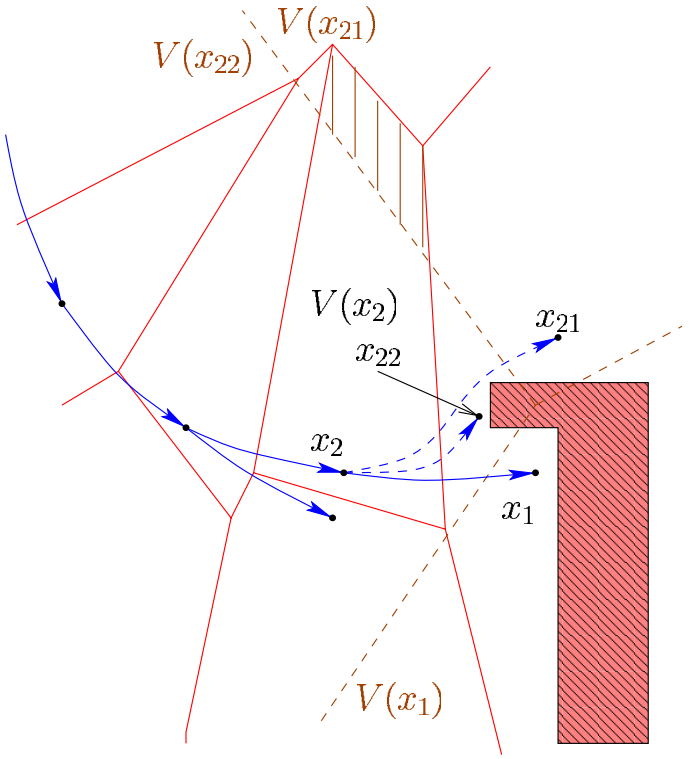


Figure 6.3: The issue in sampling the control in RRT-based planners

been applied on its state is recorded. If a control has been applied for the state of a node, it will not be considered for the node again. If the controls for a node are exhausted, this node will not be chosen for expansion any more.

Collision information: The *constraint violation probability* (CVP) of a node n is the probability of generating a trajectory in violation from state $x(n)$ with a random control constructed with sampled controls. It can be calculated by applying all possible constructed controls from state $x(n)$ and dividing the number of collision trajectories by the number of all controls. It provides global constraint information such that giving nodes with smaller CVP more priority to expand is more likely to evade obstacles. As shown in Fig. 6.4, because of differential constraints, the more close to the obstacle, the higher the CVP of the node is (darker means higher CVP). If the node with lower CVP is chosen for expansion, then the planner could have better chance to avoid the shaded obstacle region. Calculating CVP is impractical for large

control sets and number of stages (one might as well use dynamic programming to solve the problem in this case). Constraint violation tendency (CVT) is used and collected such that it will approaches but never exceed CVP such that any states with CVP less than 1 always have chance to expand.

The calculation of CVT is shown in Fig. 6.5, in which the empty leaf nodes denote trajectories in violation, the solid leaf nodes denotes violation-free trajectories, and the explored region include all reached states by the RRT. The CVT is the ratio of the number of trajectories in violation in the explored region to the number of all possible constructed trajectories, and therefore it is always an underestimation of the CVP. Each new node, n_s, in $\mathcal{T}$ is initialized with CVT at 0. Suppose that there are m controls in the sampled control set $\tilde{\mathcal{U}}_s$. The CVT of n_s is increased by $\frac{1}{m}$ if one control $\tilde{u}$ in $\tilde{\mathcal{U}}_s$ leads to a trajectory in violation. Furthermore, the CVT of its parent node, n_p, is increased by $\frac{1}{m^2}$ because 1 controls of one of its m child nodes lead to a collision. Similarly, for the k^{th} parent state of n_s, 1 collision control in node n_s will increase its CVT by $\frac{1}{m^{(k+1)}}$. This process will propagate until the root node of the RRT is reached.

Figure 6.6 shows the value of the CVT that is collected in solving an MPD problem, in which the points represent the states of nodes in the search graph and the shade of a point represents the value of the constraint violation tendency (darker indicates higher CVT). From the picture, it can be seen that these shaded points provide a good approximation of obstacles in the work environment. The closer the state of a node is to the obstacle, the higher the value of its CVT is.

The exploration information is stored as a vector at each node in the search tree. Each element of the vector corresponds to a control in $\tilde{\mathcal{U}}_s$. Each element of the vector is initially set to be *unexpanded*. If one control leads to a collision or it successfully generates a new node in $\mathcal{T}$, its corresponding element is set to be *expanded*.

The exploration information and CVT help the RRT to choose a better node

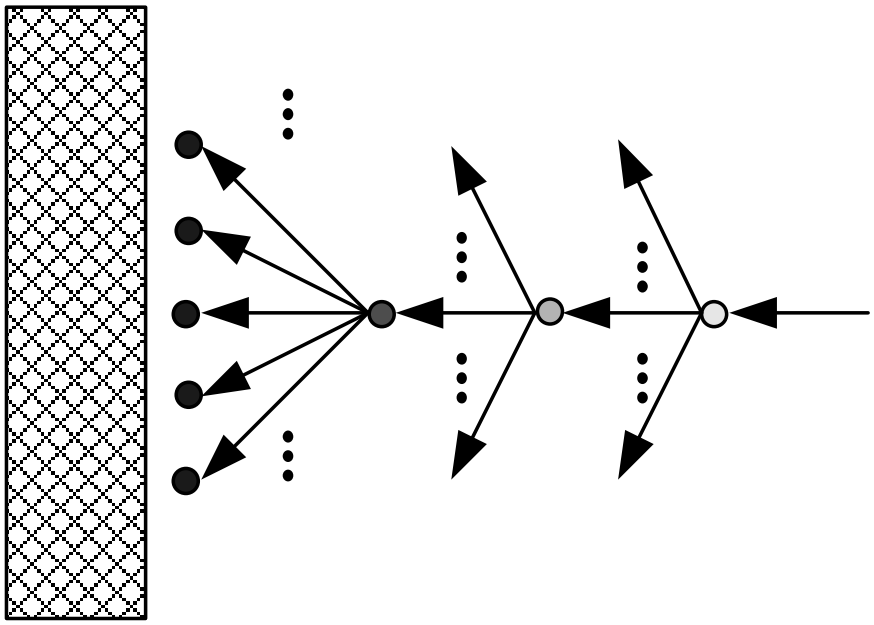


Figure 6.4: The intuition to avoid obstacle with CVP

n_{near} for exploration. In node selection, whether all controls of a node n have been expanded will be checked first. If they are all expanded, node n is ignored; otherwise, the probability of not choosing node n equals to the CVT of x. If none of the nodes is selected, one node with unexpanded controls will be chosen.

The exploration information is also helpful for sampling the controls to generate new trajectories. When node n_{near} is selected, whether a control $\tilde{u}$ is *expanded* will be checked. If it is *expanded*, it will not be considered; otherwise, it will be applied to generate a new trajectory from state $x(n_{near})$ to final state x'_{new}. If the new trajectory is violation-free and state x'_{new} has the lest distance to x_{rand} over all final states from sampled controls, x'_{new} will be associated with a new node in $\mathcal{T}$; otherwise, constraint information will be collected.

The improved RRT can be incorporated into any of the RRT-based planners described in [55]. A bidirectional planner expands two trees from both x_{init} and x_{goal}. Its performance is generally better in comparison to a single-tree planner. The original RRT only checks if two x_{new} from both trees are closed enough for a solution.

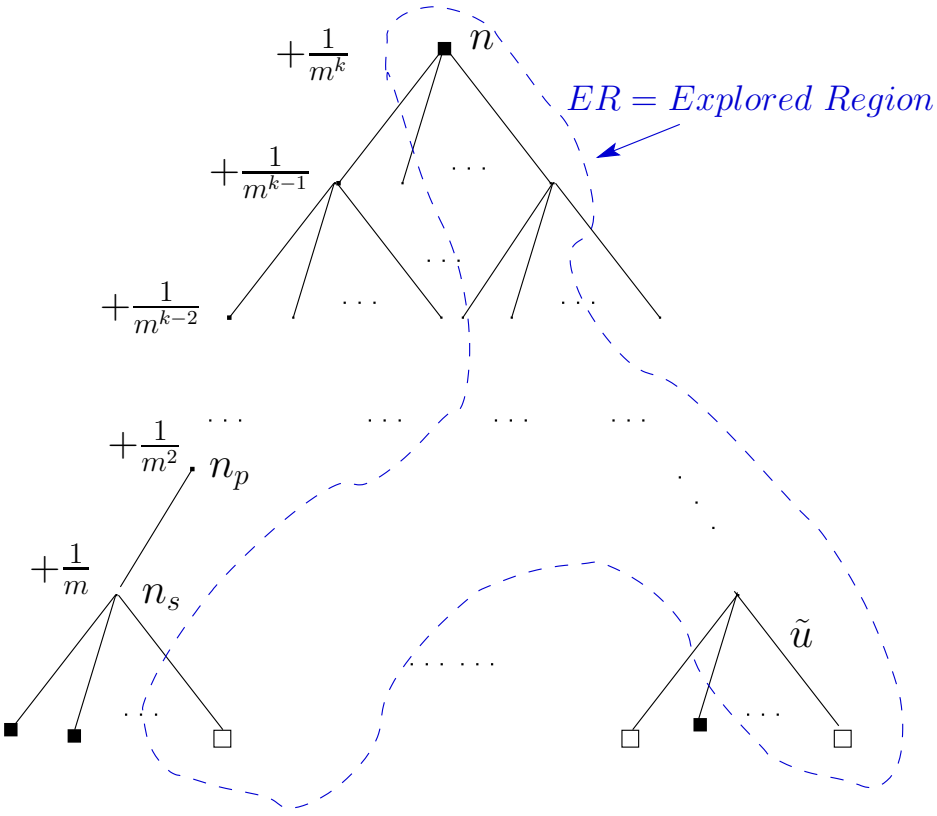


Figure 6.5: Estimation of CVP with CVT

Because the metric problem, it is possible that the original planner might continue to explore the state space even there exists a solution. The exploration frontiers of two search trees passing through each other was also mentioned in [86]. To overcome the above problem, whether each new node in one tree is within a specified distance to any node in the other tree is test. Although costly, it generally leads to reliable performance because all alternatives are considered.

6.3 Simulation Results

Our implementation was built on top of the C++ Motion Strategy Library. Experiments were conducted on a 1GHz PC running Red Hat Linux 6.2. In lane changing

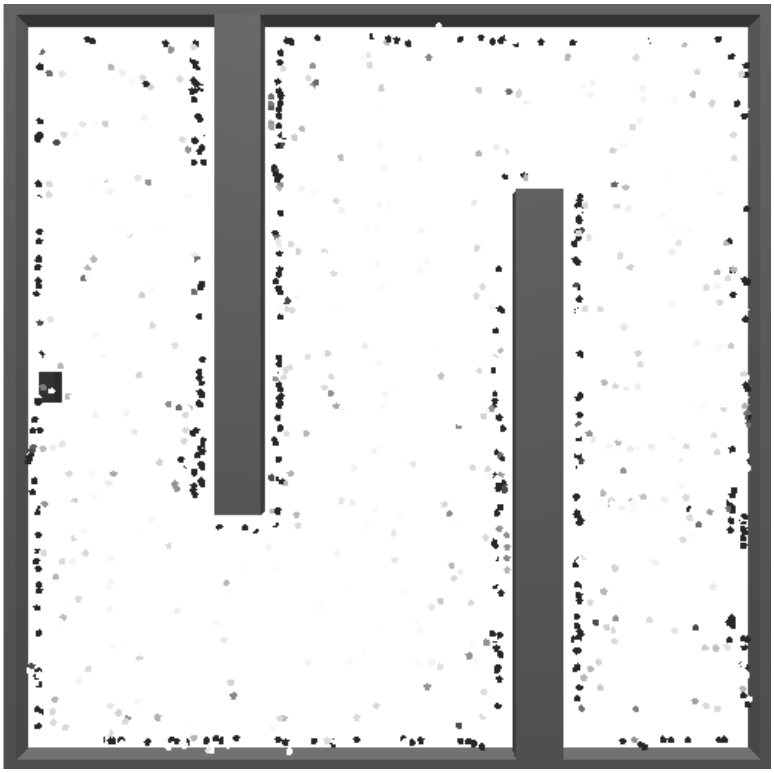


Figure 6.6: Display of CVT that is collected in a planning process

experiments, comparisons between the original RRT and the improved RRT are done. Several challenging trajectory design experiments that include vehicle dynamics problems and spacecraft problems were used to test the performance of the new method. In all the simulations, a simple weighted Euclidean metric is used in each planners.

6.3.1 System models

A 9-dimensional car model The model adapted from [87] is used in the simulation. Figures 6.7 and 6.8 show the parameters, in which WC subscript denotes an inertial frame fixed in the work environment and LC subscript denotes a local frame fixed on the car. The yaw angle of the car is ψ that corresponds to the rotation around z of WC. The velocity of yaw angle of the car is r. The roll angle of the car is ϕ that describes the rotation around x of LC. The velocity of the roll angle is q. The sideways speed (arising from slipping) is ν that is with respect to the y in LC.

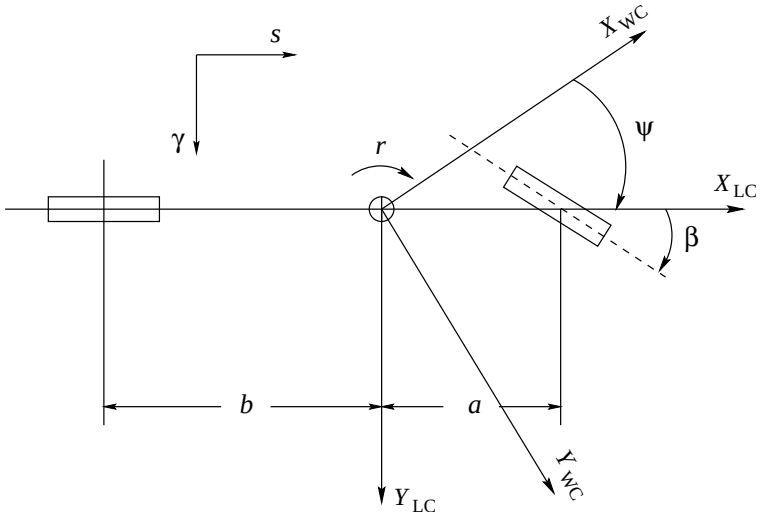


Figure 6.7: Top view of the car model

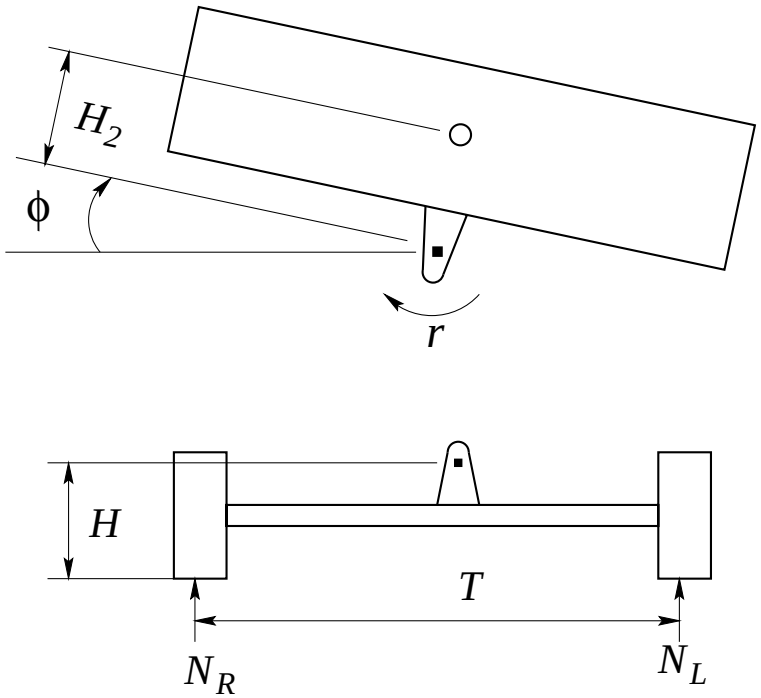


Figure 6.8: Front view of the car model

The steering angle β is with respect to z of LC. The forward speed s of the car that is with respect to x of LC. Constant $C_{\alpha f}$ and $C_{\alpha r}$ are the total cornering stiffness of the front and rear wheels, respectively. The car mass is M. The yaw moment of inertia of the car is I_y. The roll moment of inertia is I_r. The distance from the center of the front and rear axles to the car mass center, respectively, with respect to LC is a and b. The height of the joint connecting the chassis with the car frame is H. The chassis and frame are flexibly attached to model a simple suspension system. The distance between the left and right wheels is T. The distance from the joint to the mass center of the car frame is H_2. Constants K, c, and μ are described in [87].

A state of the system is represented by

$$[x, y, r, \psi, \phi, q, \nu, s, \beta]^T. \tag{6.1}$$

The system has two inputs,

$$u = [u_1, u_2]^T, \tag{6.2}$$

in which u_1 determines the changing rate of the forward speed s and u_2 determines the changing rate of the steering angle β.

The slipping angle of the front and rear wheels, denoted as α_f and α_r respectively, with respect to z of LC are expressed as

$$\alpha_f = \frac{\nu + ar}{s} - \beta \tag{6.3}$$

and

$$\alpha_r = \frac{\nu - br}{s}. \tag{6.4}$$

Because of the rolling effect, the load on the wheels are different. Let N_{fl}, N_{fr}, N_{rl}, and 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. If the friction forces are not enough, it is possible for the car to slip sideways. The y-direction forces on four wheels are:

$$F_{yfl} = -C_{\alpha f}\alpha_f/2, \ \textit{if } N_{fl}\mu > C_{\alpha f}\tan(|\alpha_f|)/2 \tag{6.5}$$

$$F_{yfr} = -C_{\alpha f}\alpha_f/2, \ \ if\ N_{fr}\mu > C_{\alpha f}\tan(|\alpha_f|)/2 \tag{6.6}$$

$$F_{yrl} = -C_{\alpha r}\alpha_r/2, \ \ if\ N_{rl}\mu > C_{\alpha r}\tan(|\alpha_r|)/2 \tag{6.7}$$

$$F_{yrr} = -C_{\alpha r}\alpha_r/2, \ \ if\ N_{rr}\mu/2 > C_{\alpha r}\tan(|\alpha_r|)/2\ , \tag{6.8}$$

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

$$F_{yfl} = -\mu N_{fl}\mathrm{Sgn}(\alpha_f)(1 - x_{fl}/2), x_{fl} = N_{fl}\mu C_{\alpha f}\tan(|\alpha_f|)/2 \tag{6.9}$$

$$F_{yfr} = -\mu N_{fr}\mathrm{Sgn}(\alpha_f)(1 - x_{fr}/2), x_{fr} = N_{fr}\mu C_{\alpha f}\tan(|\alpha_f|)/2 \tag{6.10}$$

$$F_{yrl} = -\mu N_{rl}\mathrm{Sgn}(\alpha_r)(1 - x_{rl}/2), x_{rl} = N_{fl}\mu C_{\alpha r}\tan(|\alpha_r|)/2 \tag{6.11}$$

$$F_{yrr} = -\mu N_{rr}\mathrm{Sgn}(\alpha_r)(1 - x_{rr}/2), x_{rr} = N_{fr}\mu C_{\alpha r}\tan(|\alpha_r|)/2. \tag{6.12}$$

Let

$$F_{yf} = F_{yfl} + F_{yfr}, \tag{6.13}$$

$$F_{yr} = F_{yrl} + F_{yrr}, \tag{6.14}$$

$$h = (-(K - MgH_2)\phi - cq - (F_{yf} + F_{yr})H_2)/I_r, \tag{6.15}$$

and

$$\lambda = \frac{b}{a+b}. \tag{6.16}$$

The difference between the left wheels and right wheels is

$$N_{dif} = \frac{(-K\phi - Cq + (F_{yf} + F_{yr})H)}{T/2}, \tag{6.17}$$

and current load on different wheels is changed to be

$$N_{fl} = 9.8M\lambda/2 + N_{dif}\lambda/2 \tag{6.18}$$

$$N_{fr} = 9.8M\lambda/2 - N_{dif}\lambda/2 \tag{6.19}$$

$$N_{rl} = 9.8M(1-\lambda)/2 + N_{dif}(1-\lambda)/2 \tag{6.20}$$

$$N_{rr} = 9.8M(1-\lambda)/2 - N_{dif}(1-\lambda)/2. \tag{6.21}$$

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

$$\begin{cases} \dot{x} &= s\cos\psi - \nu\sin\psi \\ \dot{y} &= s\sin\psi + \nu\cos\psi \\ \dot{r} &= (F_{yf}a - F_{yr}b)/I_y \\ \dot{\psi} &= r \\ \dot{\phi} &= q \\ \dot{q} &= h \\ \dot{\nu} &= (F_{yf} + F_{yr})/M - sr - H_2 h \\ \dot{s} &= u_1 \\ \dot{\beta} &= u_2 \end{cases} . \tag{6.22}$$

A 5-dimensional simplified car model If the rolling effect is ignored and the forward speed is fixed, the simplified model can be derived from the 9-dimensional car model. A state of the car is represented by

$$[x, y, r, \psi, \nu]^T. \tag{6.23}$$

The system only has one input u_1 which determines the steering angle.

The slipping angle of the front and rear wheels are respectively

$$\alpha_f = \frac{\nu + ar}{s} - u_1 \tag{6.24}$$

and

$$\alpha_r = \frac{\nu - br}{s}. \tag{6.25}$$

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

$$F_{yf} = -C_{\alpha f}\alpha_f, \ \textit{if } N_f\mu/2 > C_{\alpha f}\tan(|\alpha_f|) \tag{6.26}$$

and

$$F_{yr} = -C_{\alpha r}\alpha_r, \ \ if\ N_r\mu/2 > C_{\alpha r}\tan(|\alpha_r|)\ . \tag{6.27}$$

Otherwise,

$$F_{yf} = \mu N_f \mathrm{Sgn}(\alpha_f)(1 - x_f/2), x_f = N_f\mu/2C_{\alpha f}\tan(|\alpha_f|) \tag{6.28}$$

and

$$F_{yr} = \mu N_r \mathrm{Sgn}(\alpha_r)(1 - x_r/2), x_r = N_f\mu/2C_{\alpha r}\tan(|\alpha_r|). \tag{6.29}$$

Let

$$h = (-(K - MgH_2)\phi - cq - (F_{yf} + F_{yr})H_2)/I_r. \tag{6.30}$$

The following represent the motion equations:

$$\begin{cases} \dot{x} &= s\cos\psi - \nu\sin\psi \\ \dot{y} &= s\sin\psi + \nu\cos\psi \\ \dot{r} &= (F_{yf}a - F_{yr}b)/I_y \\ \dot{\psi} &= r \\ \dot{\nu} &= (F_{yf} + F_{yr})/M - sr \end{cases} \tag{6.31}$$

A 12-dimensional underactuated spacecraft model The spacecraft shown in Fig. 6.9 can translate and rotate in 3D space by firing three thrusters. These thrusters provide the driving forces and torques because the force direction does not pass through the mass center. The state of the system is represented by (g, ξ), in which

$$g = (p, R) \in SE(3) \tag{6.32}$$

represents position, p in $\mathbb{R}^3$ and orientation, R in $SO(3)$, and

$$\xi = (\hat{\Omega}, V) \in se(3) \tag{6.33}$$

denote the translational and rotational velocity expressed in the body frame. The skew matrix $\hat{\Omega}$ is defined as unique matrix for which

$$\hat{\Omega}v = \Omega \times v, \forall v \in \mathbb{R}^3. \tag{6.34}$$

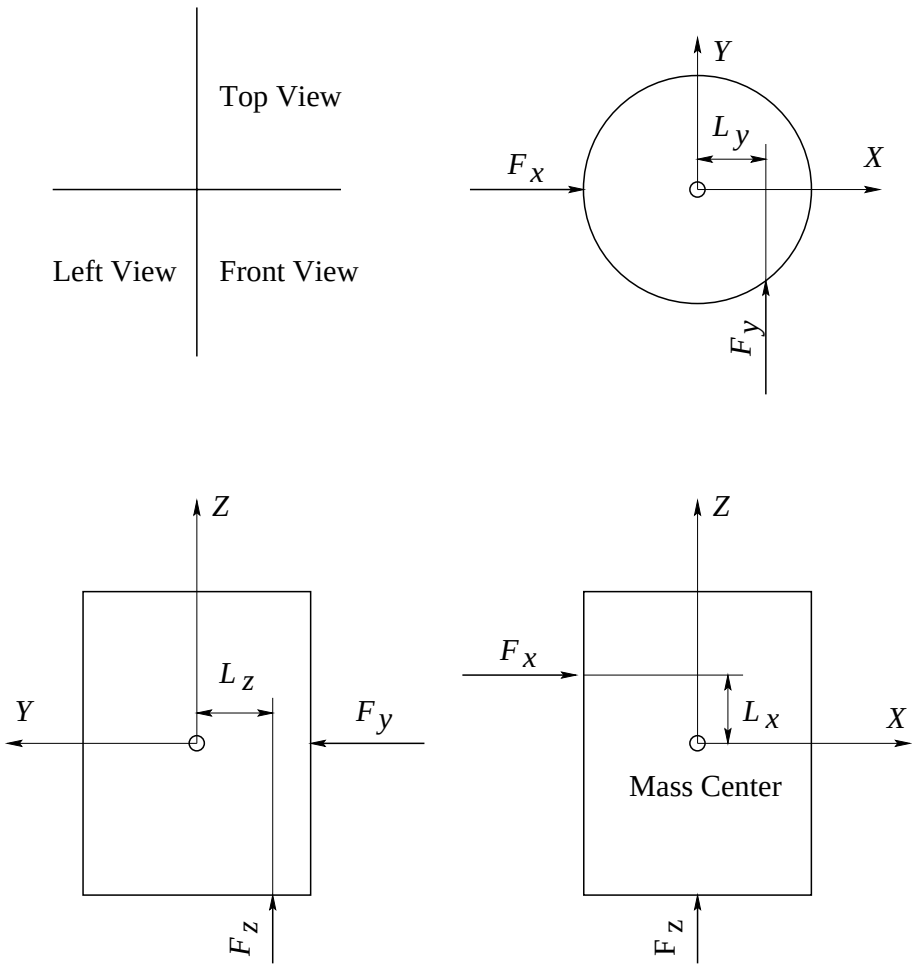


Figure 6.9: The sketch of the forces from thrusters on the spacecraft

The inputs are u_1, u_2, and u_3, which are the signed magnitude of the forces F_x, F_y, and F_z. The motion equations are:

$$\begin{cases} \dot{R} &= R\hat{\Omega}, \\ \dot{p} &= RV, \\ \dot{\Omega} &= -J^{-1}(\Omega \times (J\Omega) - f_\Omega), \\ \dot{V} &= V \times \Omega + f_V/M, \end{cases} \tag{6.35}$$

in which J is the inertial matrix, M is the mass,

$$f_V = [u_0, u_1, u_2]^T \tag{6.36}$$

denotes the signed magnitude of translational forces F_x, F_y, and F_z,

$$f_\Omega = [-u_2 L_z, u_0 L_x, u_1 L_y]^T \tag{6.37}$$

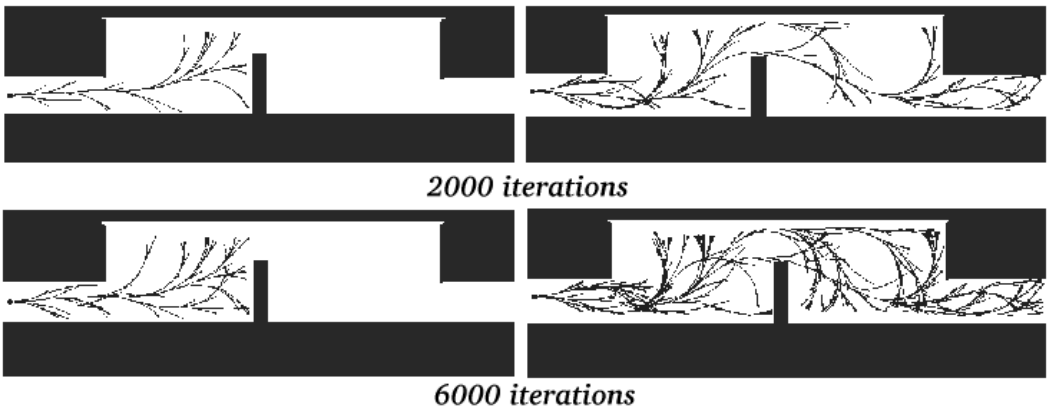


Figure 6.10: Comparison of explorations of the original RRT and improved RRT

denotes the signed magnitude of rotational torques, and L_x, L_y, and L_z are vertical distance from the mass center to the direction of forces generated by thrusters.

6.3.2 Simulation results

The lane change problem The improved planner was used to solve the problem in Fig. 2.9, in which the 5-dimensional car in Eq. (6.31) drives at 96kph to complete a lane changing maneuver in a 305m stretch of road. This problem is referred to in the automotive industry as the Consumer Union Short Course. Explorations of the original RRT and improved RRT after 2000 and 6000 iterations are compared in Fig. 6.10. Exploration of the original RRT is limited because some nodes are repeatedly selected for expansion without making progress.

Table 6.1 gives some comparative statistics for solutions to the lane changing problem under the application of the bidirectional planner. Fifty trials were performed in which six versions were run: the original and improved RRT-based planners respectively with 2000, 4000, and 8000 iterations. Note that the success rate improves dramatically. Furthermore, the average number of nodes generated by the improved RRT is greatly increased, indicating greater exploration. Also, less collision detection was performed by the improved RRT. Although computation times are comparable, note that most of the original RRT trials result in failure.

	RRT Planner			Improved RRT Planner		
Iter.	2000	4000	8000	2000	4000	8000
Suc. Rate	1/50	0/50	4/50	23/50	37/50	49/50
Num. Node	336	-	636.5	1359	2542	3283
Num. Col.	56.9	-	27.3	2.48	4.49	5.75
T(second)	10.61	-	52.15	11.9	28.8	44.7

Table 6.1: Comparison of the original and improved RRT-based planners, in which "It" means how many iterations the planners have run, "Suc. Rate" means the number of successes out of 50 trial, "Num. Node" means the average number of nodes in the search graph, "Num. Col." means the average number (in thousands) of collision checking , "T" means the average time to find the solution

The virtual driving problem The single-directional RRT-based planner was used to solve the virtual driving problem in Fig. 1.13. The model is the 9-dimensional as shown in Eq. (6.22). If the pressure on one tire is less than 0, the car is in a dangerous state. This makes it very difficult to control. Both the original and improved planners tried on this problem for 50 trials, in each of which there are $60,000$ RRT iterations. The improved RRT planner found the solution 38 times with an average of 989.65 seconds and 25712.1 nodes. The original RRT-based planner performed much worse, finding only 10 solutions out of 50 trials (note that either success rate can be improved by increasing the number of iterations).

Trajectory design for the spacecraft One simulation involves moving the underactuated spacecraft out of a cage by firing thrusters (Fig. 6.11). For 50 trials and $40,000$ RRT iterations in each trial, the improved bi-directional RRT planner solves the problem 41 times with an average of 737.32 seconds and $17,129$ nodes. The original bidirectional RRT planner only solved the problem 3 times out of 50 trials, even though $100,000$ RRT iterations were used in each trial.

In the final simulation, the improved bi-directional planner was used to solve the

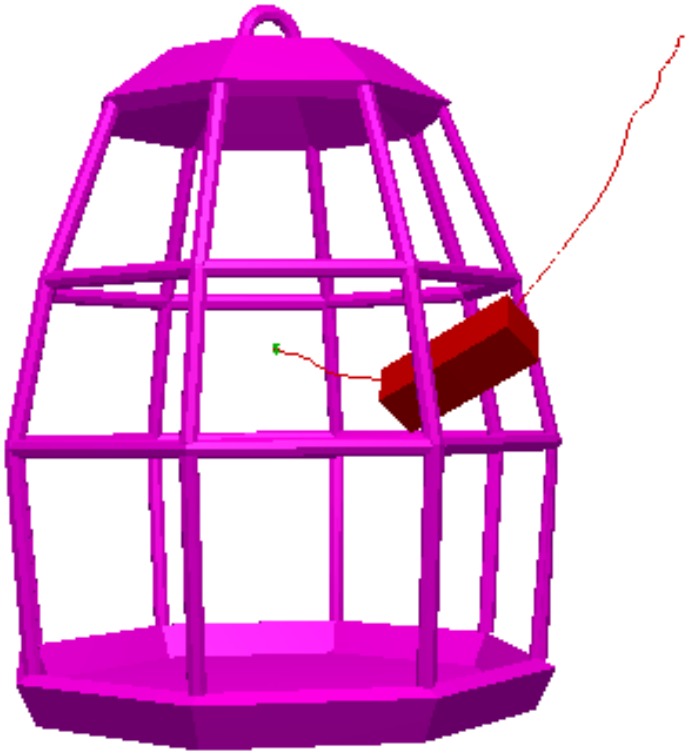


Figure 6.11: A solution of firing thrusters to move a spacecraft out of a cage

extremely challenging problem in Fig. 1.14, in which the underactuated spacecraft needs to move from one corner of the 3D-grid to another corner. The planner was run on the problem for 50 trials. In each trial, the algorithm will keep running until all node-edge pairs in the search graph are explored. The improved planner solved the problem fifty times with an average of 8059.31 seconds. The original planner did not find a solution after running for 48 hours.

Chapter 7

Conclusion

7.1 Summary

In this thesis, three problems related to sampling-based MPD algorithms have been solved. Chapter 4 discusses resolution completeness of these algorithms. The concept of resolution completeness is extended from sampling-based planners for path planning problems to those for MPD problems. The resolution completeness analysis is based on the relationship between the reachability graph and the search graph. The reachability graph is an intrinsic graph representation of a given MPD problem, while the search graph is constructed by the planner to explore the reachability graph. Because of sampling in the state space and control space, state mismatches and control mismatches could be induced. A resolution complete planner will find an existing solution or its approximation in finite time despite of these mismatches. Furthermore, with Lipschitz conditions on the motion equations and local planners, quantitative conditions on parameters of the planners are provided.

Chapter 5 solves the gap problem for MPD planners. Because of sampling, gaps could exist in the solution path. Since these gaps greatly degrade the quality of the returned solutions, small gap tolerance is normally desired, which dramatically increases the running time. The problem is solved by combining the existing planners

with a gap reduction algorithm for a class of systems with symmetry. A solution path with small gap tolerance could be quickly returned by efficiently reducing the big gaps in solution path candidates. The gap reduction algorithm can be considered an optimization of the distance between the two end points of the gap. Symmetry structures of robotic systems are employed to accelerate the optimization process by avoiding unnecessary numerical integrations. The gap reduction algorithm has been combined with single- and bi-directional planners and a PRM-based planner. Substantial performance improvements over the original planners have been observed.

Chapter 6 solves the metric sensitivity problem in RRT-based planners. RRT-based planners achieve fast exploration of the search space for path planning problems using simple Euclidean metrics. However, under differential constraints, the simple metrics have difficulties to guide the system to avoid the local minima. One ways is to calculate a good metric, which is normally as difficult as solving the MPD problem. The method in this thesis collects collision information during the search process and uses collected information to guide the search. These information is stored as a real number in each node in the search graph. The higher the number of a node is, the higher is the number of trajectories that are from the state of the node and have been detected in collision. Giving nodes with lower number higher probability to extend tends to have larger chance to avoid obstacles. The results of the simulations with these improved planners verified the effectiveness of method.

7.2 Future Directions

Even though path planning problems have been well understood through recent research, motion planning with differential constraints is still in a primitive stage. The following lists many directions, open problems, and possible improvements over existing methods and applications.

1. In theoretical analysis, exact algorithms for general MPD problems are still unknown. It seems that there does not exist a unified search space as a counterpart of configuration space for the path planning problem.

2. Motion equations characterize the evolution of the robotic system in MPD problems. These equations could be derived using intrinsic geometric structures of the systems when the configuration space has no singularities. It is desirable to model the system with singularities, such as the system with closed chains. Furthermore, the properties of motion equations are desirable, which will tell the mobile ability and limit of the robotic system. For example, the nonslipping model of the roller racer [88] is proved that it cannot be stopped only by the steering angle control if it starts moving, which tells that it is impossible to design a trajectory for the model to move from one configuration at rest to a different configuration at rest.

3. For practical algorithms, understanding of complexity of sampling-based algorithms is desired. The current resolution completeness analysis of sampling-based algorithms is limited to the system with only one set of motion equations. It is desirable to extend the resolution complete conditions for MPD problems with hybrid systems, which are describe by multiple sets of motion equations. Also less conservative resolution completeness conditions should be explored using structures of specific systems.

4. MPD algorithms have a large amount of potential applications. Realistic animation in computer graphics has strong similarity with MPD problems. Since the motion equation is derived from the physical laws, the trajectory of a control could be used to generate a realistic animation of the system and MPD algorithms could be used to automatically generate animations. System verification has attracted more and more attention in many areas, in which safety

has high priority. MPD algorithms could be used as a falsification tool to find whether an unsafe state could be reached, and resolution completeness analysis could provide some safety guarantee.

Appendix A

Appendix

In the appendix, existence and requirements of asymptotic finite sampling in the state space and control space are provided. Resolution complete conditions for planners using nonconnecting local planners and numerical calculations are also provided.

The asymptotic state space sampling could be easily achieved though sampling with discretization. In planners using connecting local planners, the asymptotic control space sampling could be identified with a state space sampling with discretization. However, when a nonconnecting local planner is used, the asymptotic control space sampling is not trivial, which is one of the main question solved in this appendix.

As described in Section 3.3.2, the control space sampling in planners using nonconnecting local planners could be implemented through time sampling and input space sampling. Therefore, an asymptotic finite control space sampling needs both the asymptotic finite time sampling and the asymptotic finite input space sampling. The asymptotic finite time sampling could be similarly constructed to the asymptotic finite state space sampling. In the first section, an asymptotic finite input space sampling is constructed for a simple sampling control set, in which each control is a continuous function from a fixed time interval to a one-dimensional input space and has an upper bound on the magnitude of its first-order time derivative, i.e., the controls have a slope bound.

In the second section, it is shown that the slope bound of the sampling control set is necessary for the asymptotic finite input space sampling. Existence of asymptotic finite sampling for general state space and control space is respectively shown by construction in the third section.

A.1 An Asymptotic Finite Input Space Sampling in a Simple Sampling Control Set with a Slope Bound

To present the construction of the sampling clearly, a simple sampling control set $\tilde{\mathcal{U}}$ is assumed to be a *restricted function space*, denoted as $\mathcal{F}_T$. Each element of $\mathcal{F}_T$ is a continuous and piecewise first-order differentiable function from $[0, T]$ to a one-dimensional input space

$$[u_{\min}, u_{\max}] \subset (-\infty, \infty), \tag{A.1}$$

and the magnitude of its first order time derivative is no larger than a positive real number D_p. The objective of this section is to construct an asymptotic finite input space sampling in $\mathcal{F}_T$ such that for any given positive ϵ, the sampling will generate a finite sampled control set $\mathcal{F}_s$ whose dispersion with respect to $\mathcal{F}_T$ and the infinity norm is less than ϵ. The construction in this section could be easily extended for cases in which the input space is m-dimensional.

In the following, the sampling will be first described to generate a finite sampled control set for a given dispersion bound, and then the dispersion of the sampled control set will be shown to be less than the given dispersion bound.

Given a dispersion bound ϵ, the construction (sketched in Fig. A.1) of a finite sampled control set for an asymptotic finite sampling in $\mathcal{F}_T$ is as follows:

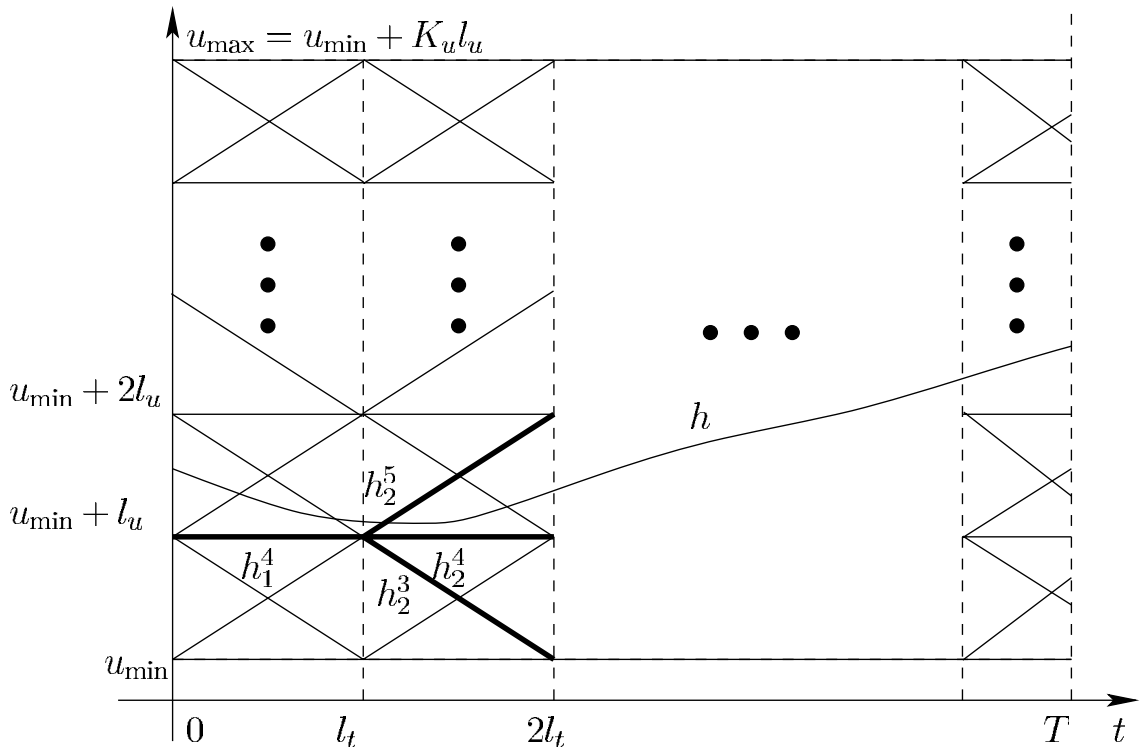


Figure A.1: Asymptotic finite sampling in the simple sampling control set, i.e., a restricted function space

1. Choose

$$0 < l_u < \epsilon \tag{A.2}$$

such that

$$K_u = \frac{|u_{max} - u_{min}|}{l_u} \tag{A.3}$$

is a positive integer. Then choose

$$l_t = \frac{l_u}{D_p}. \tag{A.4}$$

Let

$$K_t = \lfloor \frac{T}{l_t} \rfloor. \tag{A.5}$$

2. Construction of sampled controls on interval $[kl_t, (k+1)l_t]$, in which $k = 0, 1, \cdots, K_t - 1$. In each intervals, these sampled controls are linear functions that start from point set

$$U_s = \{u_{\min}, u_{\min} + l_u, \cdots, u_{\min} + K_u l_u\} \tag{A.6}$$

with slope D_p, 0, or $-D_p$ whenever the function value remains $[u_{\min}, u_{\max}]$. For example, if a control starts from point $u_{\min} + K_u l_u$, then the control could only have slop 0 or $-D_p$. Otherwise, the value of the control could be out of the input space. By the choice of l_t, these functions also terminate in U_s.

3. Construction of sampled controls on interval $[K_t l_t, T]$. In the interval, the sampled controls are also linear functions that start from point set U_s with slope be D_p, 0, or D_p whenever function value remains in $[u_{\min}, u_{\max}]$, and stop when time T is reached.

4. Construction of the sampled control set $\mathcal{F}_s$. The set includes all controls that are constructed by continuously combining partially defined controls in Step 2 and 3. For example, in Fig. A.1, the control h_1^4 partially defined on $[0, l_t]$ is connected to h_2^3, h_2^4, and h_2^5 partially defined on $[l_t, 2l_t]$ (These four linear functions are shown as thicker lines.) to obtain three functions on $[0, 2l_t]$. Similarly, sampled controls defined on $[0, T]$ could be constructed.

Lemma 18 *The set $\mathcal{F}_s$ is finite and its dispersion with respect to $\mathcal{F}_T$ and the infinity-norm is less than ϵ.*

Proof: Since there are K_u starting points and at most three slopes, the number of linear functions defined on each l_t interval or $[K_t l_t, T]$ is at most $3K_u$. Each linear function could be connected to at most 3 linear functions in the next interval. Since there are only $K_t + 1$ intervals, the number of functions in $\mathcal{F}_s$ is less than $3^{K_t} K_u$.

To check the dispersion is less than ϵ, it is enough to show that for any function h in $\mathcal{F}_T$, there always exist h' in $\mathcal{F}_s$ such that

$$||h - h'||_\infty \leq \epsilon. \tag{A.7}$$

Function h is firstly extended with constant value $h(T)$ to interval $[T, (K_t + 1)l_t]$ to obtain h''. Let h_i for $i = 0, 1, \cdots, K_t + 1$ be the value of h'' at time $0, l_t, \cdots,$

$(K_t + 1)l_t$. Assume that a_0 be the nearest point in U_s to h_0. If a_0 has two choices, it could be either one. Similarly, we could have a_{k+1} be the nearest point in U_s to h_{k+1} for $k = 0, 1, \cdots, K_t$. If a_{k+1} has two choices, it chooses the one which is closer to a_k. Because of derivative bound of h',

$$|a_{k+1} - a_k| \leq l_u \tag{A.8}$$

by the choice of l_t and l_u. Let g is the function which composes of linear functions connecting $a_0, a_1, \cdots, a_{K_t}$. It can be verified that on any l_t interval,

$$||g - h''|| \leq l_u \tag{A.9}$$

since

$$|h_i - a_i| \leq \frac{l_u}{2} \tag{A.10}$$

for any $i = 0, 1, \cdots, K_t + 1$ and

$$D_p l_t = l_u. \tag{A.11}$$

Choosing h' to equal g on interval $[0, T]$ completes the proof. □

A.2 Nonexistence of Asymptotic Finite Sampling in a Sampling Control Set without a Slope Bound

In this section, it is shown that there does not exist an asymptotic control space sampling if there is no slope bound on the controls in the sampling control set $\bar{\mathcal{U}}$, i.e., $\bar{\mathcal{U}}$ for planners using nonconnecting local planners, does not satisfy Assumption 1 in Chapter 4. The nonexistence needs the following lemma, which could be used to show that the dispersion of a finite sampled control set from any input space sampling can not be arbitrarily small.

Lemma 19 *Assume that a sampling control set $\bar{\mathcal{U}}$ does not satisfy Assumption 1. For any finite sampled control set*

$$\mathcal{U}_B = \{\tilde{u}_1, \tilde{u}_2, \cdots, \tilde{u}_l\} \subset \bar{\mathcal{U}}, \tag{A.12}$$

there always exist a positive real number a and control $\tilde{u}_s$ in $\bar{\mathcal{U}}$ such that for any $\tilde{u}'$ in $\mathcal{U}_B$,

$$\|\tilde{u}_s - \tilde{u}'\|_\infty \geq a. \tag{A.13}$$

Proof: The real number a and control $\tilde{u}_s$ are constructed in two steps. In the first step, a and $\tilde{u}_s$ are constructed such that

$$\|\tilde{u}_s - \tilde{u}_j\|_\infty > a \tag{A.14}$$

for one $\tilde{u}_j$ in $\mathcal{U}_B$. In the second step, $\tilde{u}_s$ is constructed such that Eq. (A.14) is satisfied for all controls in $\mathcal{U}_B$.

Construction of a and $\tilde{u}_s$ with respect for one $\tilde{u}_j$ in $\mathcal{U}_B$ Each coordinate $\tilde{u}_s^i$ of

$$\tilde{u}_s = [\tilde{u}_s^1, \tilde{u}_s^2, \cdots, \tilde{u}_s^m]^T \tag{A.15}$$

is constructed with respect to some t in $(0, \bar{t}(\tilde{u}_j))$ as shown in Fig. A.2. Since t is an interior point such that there is a Δt neighborhood of t for some positive real number Δt. The function $\tilde{u}_s^i$ will be the same as $\tilde{u}_j^i$ except on interval $[t - \Delta t, t + \Delta t]$. If

$$\tilde{u}_j^i(t) \geq \bar{u}^i \tag{A.16}$$

with

$$\bar{u}^i = \frac{u_{\min}^i + u_{\max}^i}{2}, \tag{A.17}$$

then $\tilde{u}_s^i$ is defined as straight segments connecting $\tilde{u}_j^i(t - \Delta t)$, $u_{\min}^i$, and $\tilde{u}_j^i(t + \Delta t)$ on $[t - \Delta t, t + \Delta t]$; otherwise, $\tilde{u}_s^i$ consists of straight segments connecting $\tilde{u}_j^i(t - \Delta t)$, $u_{\max}^i$, and $\tilde{u}_j^i(t + \Delta t)$ as shown in Fig. A.2. Choosing

$$a = \|[\bar{u}^1, \bar{u}^2, \cdots, \bar{u}^m]^T\|, \tag{A.18}$$

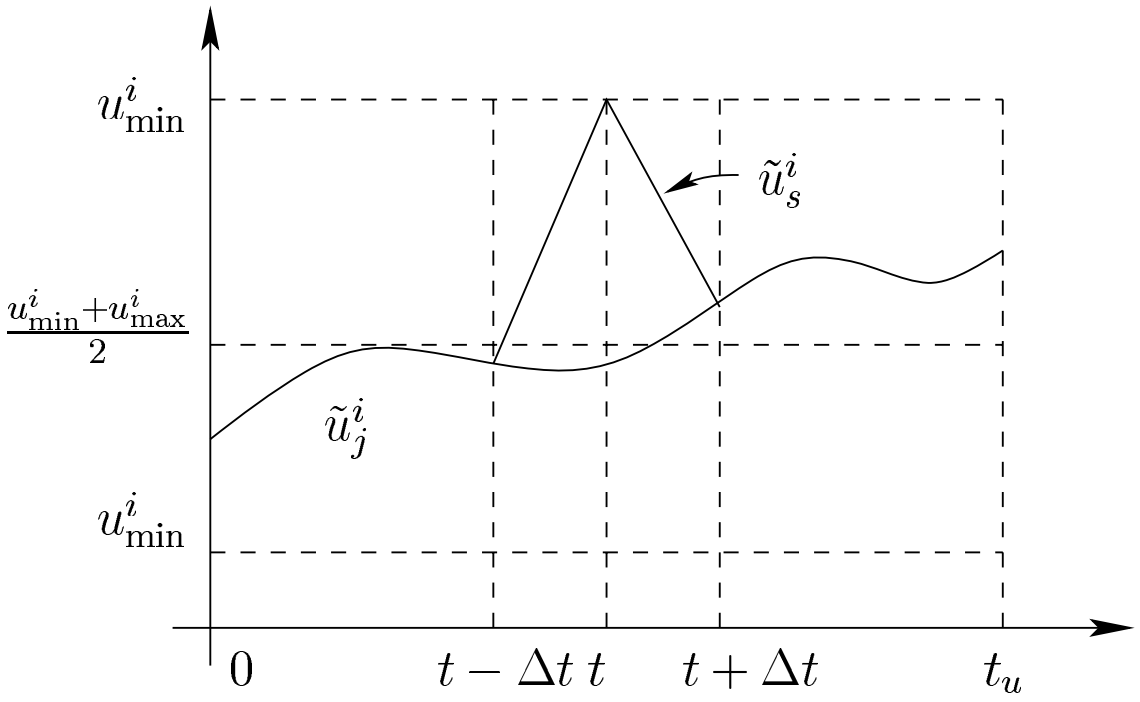


Figure A.2: The construction of $\tilde{u}^i_s$ for some $\tilde{u}^i_j$ in $\mathcal{U}_B$

it is easy to see that

$$\|\tilde{u}_s - \tilde{u}_j\|_\infty \geq a. \tag{A.19}$$

Construction of $\tilde{u}_s$ with respect for all controls in $\mathcal{U}_B$ Assume that all controls in $\mathcal{U}_B$ share a common domain $[0, t_c]$, in which we can choose l different interior points

$$\{p_1, p_2, \cdots, p_l\}. \tag{A.20}$$

Then, some positive real number Δt is chosen such that the Δt neighborhoods of these l points are in $[0, t_c]$ and are disjointed.

On the neighborhood of the i-th point, $\tilde{u}_s$ is constructed with respect to $\tilde{u}_i$ as shown in the previous. For intervals outside of these neighborhoods, $\tilde{u}_s$ is any vector-valued functions such that $\tilde{u}_s$ is continuous, which can be achieved by connecting adjacent boundary values at each neighborhood by straight segments.

It is easy to see that

$$\|\tilde{u}_s - \tilde{u}_j\|_\infty > a \tag{A.21}$$

for any $\tilde{u}_j$ in $\mathcal{U}_B$. $\square$

A.3 Existence of Asymptotic Finite Sampling

Asymptotic finite sampling is a very important assumption for resolution complete planners. In this section, it is shown that such sampling does exist by construction. These sampling could be used in planners to achieve resolution completeness. Note that the constructed asymptotic sampling in this section is used to show its existence. There could also exist other asymptotic finite sampling.

A.3.1 An asymptotic finite state space sampling

One such sampling is achieved by combing a state space sampling with an implicit discretization as follows. Given any positive real number α, a new node will be added to G when the state of the node is not in the α neighborhood of any existing states in X_G with respect to the given norm on X.

Lemma 20 *The above state space sampling with discretization is an asymptotic finite state space sampling, i.e., the sampling satisfies Assumption 5.*

Proof: The proof will first show that for any given positive real number α, only a finite number of nodes will be added to G. Secondly, a proper α is chosen to satisfy any given dispersion bound ϵ.

The first part is proved by contradiction. Because the state space X is a bounded set, it will be covered by a finite number of open sets

$$\{O_1, O_2, \cdots, O_l\}, \tag{A.22}$$

each of which is an open ball of radius $\frac{\alpha}{2}$. Choosing one state from each open ball obtains a finite set. Now, assume that the above state space sampling with discretization generates an infinite of nodes in the search graph G. There must exist two nodes

whose states are in a same set O_i. However, the distance between any two states in open ball of radius $\frac{\alpha}{2}$ is less than α, which is a contradiction.

The dispersion of the sampling is measured by the dispersion of the maximal sampled set X_s with respect to state space X. A maximal sampled set X_s is constructed by keeping inserting a new state that is not in the α neighborhood of any state in the current X_s until no new state could be inserted, i.e., the neighborhoods of states in X_s cover the state space X. Therefore, for any positive real number ϵ, choosing

$$\alpha < \frac{\epsilon}{2} \tag{A.23}$$

will make the finite sampled state set have dispersion less than ϵ; otherwise, there exists an empty ball of radius ϵ which cannot be covered by open balls of radius

$$\alpha < \frac{\epsilon}{2} \tag{A.24}$$

with their centers outside the empty ball. □

A.3.2 An asymptotic finite control space sampling

The existence of asymptotic finite control space sampling will be presented respectively for planners either with or without using the connecting local planners since the sampling are characterized differently based on whether a connecting local planner is used.

Asymptotic finite control space sampling for planners using connecting local planners For planners using connecting local planners which satisfy Assumption 7, control space sampling is unified with the state space sampling. The sampled control set only consists of controls that connect two sampled states. An asymptotic control space sampling is achieved by the asymptotic state space sampling in Section A.3.1.

Lemma 21 *The above unified control space and state space sampling with discretization satisfies Assumption 5, i.e., it is an asymptotic finite control space sampling.*

Proof: For any given positive real number ϵ, the state space sampling with discretization in Appendix A.3.1 could be used to obtain a finite sampled set with dispersion bound ϵ. Furthermore, since sampled control set only consists of controls between sampled states and there are only a finite number of state pairs in the sampled state set, the sampled control set $\tilde{\mathcal{U}}_s$ is finite. □

Asymptotic control space sampling in $\tilde{\mathcal{U}}$ for nonconnecting local planners Since the sampling control set $\tilde{\mathcal{U}}$ normally equals the control space generator set $\bar{\mathcal{U}}$, an asymptotic finite control space sampling will be described in the following lemma for the general $\bar{\mathcal{U}}$ that satisfies Assumption 1.

Lemma 22 *For any positive real numbers ϵ, if $\bar{\mathcal{U}}$ contains all possible continuous controls and satisfies Assumption 1, there is an asymptotic finite control space sampling in $\bar{\mathcal{U}}$ which generates a finite sampled control set with its dispersions ϵ_t and ϵ_u no larger than ϵ.*

Proof: This lemma is proved by construction. Choose real positive numbers ϵ_t and ϵ_u no larger than the given ϵ. Time sampling in $\mathcal{D} \setminus \{0\}$ of $\bar{\mathcal{U}}$ is in one dimension. It is easy to obtain a finite set

$$T_s = \{t_1, t_2, \cdots, t_k\} \tag{A.25}$$

with its dispersion no larger than ϵ_t. The important part is how to sample

$$\bar{\mathcal{U}}_t = \{\tilde{u} \in \bar{\mathcal{U}} \mid \bar{t}(\tilde{u}) = t\}, \tag{A.26}$$

for each t in T_s to obtain a finite set with its dispersion no larger than ϵ_u.

The construction starts with sampling in the set $\bar{\mathcal{U}}_{i,t}$ of i-th coordinate of controls in $\bar{\mathcal{U}}$, which is a restricted function space of functions from $[0, t]$ to $[u^i_{\min}, u^i_{\max}]$ for

some $i = 1, 2, \cdots, m$ under Assumption 1. In Appendix A.1, there is a sampling in $\bar{\mathcal{U}}_{i,t}$ that generates a finite set with arbitrarily small dispersion with respect to the infinity norm.

For a positive real number ϵ_u, there always exists a vector

$$\epsilon = [\epsilon_1 \ \epsilon_2 \ \cdots \ \epsilon_m]^T \tag{A.27}$$

with positive ϵ_i for all $i = 1, 2, \cdots, m$ such that

$$\|\epsilon\| < \epsilon_u. \tag{A.28}$$

Each $\bar{\mathcal{U}}_{i,t}$ could be sampled with dispersion bound ϵ_i to obtain a finite set $\bar{\mathcal{U}}^s_{i,t}$ using the method described in Appendix A.1. The sampled set $\bar{\mathcal{U}}^s_t$ from $\bar{\mathcal{U}}_t$ includes all controls whose coordinates are chosen from each $\bar{\mathcal{U}}^s_{i,t}$. It is clear that $\bar{\mathcal{U}}^s_t$ is finite since each $\bar{\mathcal{U}}^s_{i,t}$ is finite, and for any $\tilde{u} \in \bar{\mathcal{U}}_t$, there exists $\tilde{u}' \in \bar{\mathcal{U}}^s_t$ such that

$$\rho(\tilde{u}, \tilde{u}') \leq \|\epsilon\| < \epsilon_u. \tag{A.29}$$

Applying the same sampling for each $\bar{\mathcal{U}}_t$ will obtain a finite $\tilde{\mathcal{U}}_s$.

To check the dispersion bound, let us consider any

$$\tilde{u} = [\tilde{u}_1 \ \tilde{u}_2 \ \cdots \ \tilde{u}_m] \in \bar{\mathcal{U}}, \tag{A.30}$$

in which each $\tilde{u}_i$ is a coordinate of $\tilde{u}$. By the construction of $\tilde{\mathcal{U}}_s$, there exists

$$\tilde{u}' = [\tilde{u}'_1 \ \tilde{u}'_2 \ \cdots \ \tilde{u}'_m] \tag{A.31}$$

in

$$\bar{\mathcal{U}}^s_{t_i} \subset \tilde{\mathcal{U}}_s \tag{A.32}$$

for some t_i in T_s such that

$$|\bar{t}(\tilde{u}) - \bar{t}(\tilde{u}')| \leq \epsilon_t \tag{A.33}$$

and

$$\rho(\tilde{u}, \tilde{u}') \leq \|\epsilon\| < \epsilon_u. \tag{A.34}$$

□

A.4 Resolution Completeness Conditions for Planners using Nonconnecting Local Planners and Numerical Calculations

In MPD algorithms, trajectories of controls from states are normally calculated by integration. Since analytical integration is only available for few simple ODEs, numerical integrations are mostly used. At the same time, each number in the computer is represented by a finite number of bits, which leads to the existence of precision errors. For example, the real number π cannot be exactly represented in the computer, i.e., its stored value in the computer is different from its actual value. Both of them would generate state errors in the search graph. When state errors exist due to numerical calculations, the following resolution completeness conditions are provided for planners using nonconnecting local planners.

Theorem 23 (Conditions for planners using nonconnecting local planners and numerical calculations) *Assume that a problem $\mathcal{P}$ satisfies Assumption 1, 2 and 3. an MPD planner satisfies Assumption 4, 5 and 6, and ϵ_i and ϵ_n are the upper bounds on the numerical integration errors and precision errors. For any*

$$0 < \epsilon_p < 1, \tag{A.35}$$

if there is a K-stage solution with clearance w and tolerance ϵ_s, using state space sampling with discretization and dispersion bound ϵ_d and control space sampling with dispersion bound ϵ_u and ϵ_t, a planner will find an ϵ_p-approximation of the solution with clearance $w(1-\epsilon_p)$ and tolerance $\epsilon_s + \epsilon_p w$ in finite time under the following conditions:

$$(\epsilon_u(L_d - 1) + \epsilon_t D_f + \epsilon_d + \epsilon_i + \epsilon_n)\frac{L_d^{K+1} - 1}{L_d - 1} < w, \tag{A.36}$$

$$(\epsilon_u(L_d-1)+\epsilon_t D_f)\frac{L_d^K-1}{L_d-1} < \epsilon_p w, \tag{A.37}$$

$$\epsilon_d > \epsilon_n. \tag{A.38}$$

Proof: Assumption 2 and 6 will ensure that each stage of an approximate solution will be in the search graph.

Furthermore, with Assumption 3, Lemma 13, 15, 17, and triangular inequality, the following inequality will ensure $\hat{\tau}_\eta$ of an approximate solution with tolerance $\epsilon_s + \epsilon_p w$ exists in the search graph.

$$d_\tau(\hat{\tau}_\eta, \tau) \leq (\epsilon_u(L_d-1)+\epsilon_t D_f + \epsilon_d + \epsilon_i + \epsilon_n)\frac{L_d^{K+1}-1}{L_d-1} < w, \tag{A.39}$$

and

$$d_\tau(\tau_\eta, \tau) \leq (\epsilon_u(L_d-1)+\epsilon_t D_f)\frac{L_d^{K}-1}{L_d-1} < \epsilon_p w. \tag{A.40}$$

Lastly, $\epsilon_d > \epsilon_n$ is necessary to ensure the correct calculation. If ϵ_d is less than the bound on the precision bound ϵ_n, then unpredictable results could obtained.

The algorithm will run in finite time as shown in the proof for Theorem 9. □

References

[1] J. Bobrow, S. Dubowsky, and J. Gibson. "Time-optimal control of robotic manipulators along specified paths." *Int. J. Robot. Res.* **4** (1985).

[2] J. Hollerbach. "Dynamic scaling of manipulator trajectories." Technical report, MIT A.I. Lab Memo 700 (1983).

[3] J. Luh and C. Lin. "Optimum path planning for mechanical manipulators." *J. Dyn. Sys. Meas. Contr.* **102** 142–151 (1981).

[4] J. Luh and M. Walker. "Minimum-time along the path for a mechanical arm." In *Proc. IEEE Conf. Decision and Contr.*, pp. 755–759 (1977).

[5] K. Shin and N. McKay. "Minimum-time control of robot manipulators with geometric path constraints." *IEEE Trans. Autom. Control* **30** 531–541 (1985).

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

[7] N. J. Nilsson. "A mobile automaton: An application of artificial intelligence techniques." *Proc. Int. Joint Conf. on Artif. Intell.* pp. 509–520 (1969).

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

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

[10] 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).

[11] J. Canny. *The Complexity of Robot Motion Planning* (MIT Press, Cambridge, MA, 1988).

[12] N. Amato, O. Bayazit, L. Dale, C. Jones, and D. Vallejo. "Choosing good distance metrics and local planners for probabilistic roadmap methods." *IEEE Trans. Robot. & Autom.* **16** 442–447 (Aug 2000).

[13] S. LaValle and J. K. Jr. "Randomized kinodynamic planning." *International Journal of Robotics Research* **20** 378–400 (2001).

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

[15] E. Mazer, J. Ahuactzin, and P. Bessière. "The Ariadne's clew algorithm." *J. Artificial Intell. Res.* **9** 295–316 (November 1998).

[16] T. Simeon, J.-P. Laumond., and C. Nissoux. "Visibility based probabilistic roadmaps for motion planning." *Advanced Robotics Journal* **14** (2000).

[17] J. Yakey, S. M. LaValle, and L. E. Kavraki. "Randomized path planning for linkages with closed kinematic chains." *IEEE Transactions on Robotics and Automation* **17** 951–958 (December 2001).

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

[19] S. M. LaValle. *Planning Algorithms* ([Online], 2004). Available at http://msl.cs.uiuc.edu/planning/.

[20] G. Oriolo and Y. Nakamura. "Control of mechanical systems with second-order nonholonomic constraints." In *Proc. 30th IEEE Conference on Decision and Control*, pp. 2398–2403 (1991).

[21] J.-P. Laumond. "Feasible trajectories for mobile robots with kinematic and environment constraints." In *ECAI* (1986).

[22] B. Donald, P. Xavier, J. Canny, and J. Reif. "Kinodynamic planning." *Journal of the ACM* **40** 1048–1066 (November 1993).

[23] J. Canny, B. Donald, J. Reif, and P. Xavier. "On the complexity of kinodynamic planning." In 29^{th} *Symposium on the Foundations of Compute Science* (1988).

[24] C. ÓDúnlaing. "Motion planning with inertial constraints." *ALGO* **2** 431–475 (1987).

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

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

[28] D. Balkcom and M. Mason. "Time optimal trajectories for bounded velocity differential drive vehicles." *Int. J. Robot. Res.* **21** 199–217 (March 2002).

[29] M. Fliess, J. Levine, P. Martin, and P. Rouchon. "Flatness and defect of nonlinear systems." *International Journal of Control* **61** 1327–1361 (1995).

[30] F. Bullo and K. Lynch. "Kinematic controllability for decoupled trajectory planning in underactuated mechanical systems." *IEEE Trans. on Robotics and Automation* **17** 402–412 (2001).

[31] G. Laffierriere and H. J. Sussman. "Motion planning for controllable systems without drift." In *IEEE Int. Conf. Robot. & Autom.* (1991).

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

[33] H. Geering, L. Guzzella, S. Hepner, and C. Onder. "Time-optimal motions of robots in assembly tasks." *IEEE Trans. Automat. Contr.* **AC-31** 512–518 (June 1986).

[34] E. Meier and A. Bryson. "An efficient algorithm for time optimal control of a two-link manipulator." In *Proc. AIAA conf. Guidance Control*. Monterey, CA (1987).

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

[36] P. Cheng and S. LaValle. "Resolution complete rapidly-exploring random trees." In *IEEE Int. Conf. Robot. & Autom.* (2001).

[37] P. Choudhury and K. Lynch. "Trajectory planning for second-order underactuated mechanical systems in presence of obstacles." In *Workshop on Algorithmic Foundations of Robotics* (2002).

[38] E. Frazzoli, M. A. Dahleh, and E. Feron. "Real-time motion planning for agile autonomous vehicles." *AIAA Journal of Guidance, Control, and Dynamics* **25** 116–129 (2002).

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

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

[41] A. Marigo and A. Bicchi. "Steering driftless nonholonomic systems by control quanta." In *IEEE Conf. Decision & Control* (1998).

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

[43] 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.*, pp. 150–155. Cincinnati, OH (1990).

[44] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock. "Randomized kinodynamic motion planning with moving obstacles." In *The Fourth International Workshop on Algorithmic Foundations of Robotics* (2000).

[45] T. Karatas and F. Bullo. "Randomized searches and nonlinear programming in trajectory planning." In *IEEE Conference on Decision and Control* (2001).

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

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

[48] K. Lynch, N. Shiroma, H. Arai, and K. Tanie. "Collision free trajectory planning for a 3-dof robot with a passive joint." *Int. J. Robot. Res.* **19** 1171–1184 (2000).

[49] R. Murray and S. Sastry. "Steering nonholonomic systems using sinusoids." In *IEEE Int. Conf. Robot. & Autom.*, pp. 2097–2101 (1990).

[50] S. Sekhavat and J.-P. Laumond. "Topological property for collision-free nonholonomic motion planning: the case of sinusoidal inputs for chained form systems." *IEEE Transactions on Robotics and Automation* **14** 671–680 (1998).

[51] F. Bullo and A. Lewis. *Geometric Control of Mechanical Systems* (Springer Verlag, 2004).

[52] S. Sastry. *Nonlinear systems, analysis, stability and control* (Springer, New York, NY, 1999).

[53] Y. Chitour and B. Piccoli. "Controllability for discrete systems with a finite control set." *Math. Control Signals Syst.* **14** 173–193 (2001).

[54] A. Bicchi, A. Marigo, and B. Piccoli. "On the reachability of quantized control systems." *IEEE Trans. on Automatic Control* **47** 546–563 (April 2002).

[55] S. LaValle and J. K. Jr. "Rapidly-exploring random trees: Progress and prospects." In *2000 Workshop on the Algorithmic Foundations of Robotics* (2000).

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

[57] J. Pearl. *Heuristics* (Addison-Wesley, Reading, MA, 1984).

[58] K. Bekris, B. Chen, A. Ladd, E. Plakue, and L. Kavraki. "Multiple query probabilistic roadmap planning using single query primitives." In *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems* (2003).

[59] P. Cheng, E. Frazzoli, and S. LaValle. "Improving the performance of sampling-based planners by using a symmetry-exploiting gap reduction algorithm." In *IEEE Int. Conf. Robot. & Autom.* (2004).

[60] Z. Shiller and S. Dubowsky. "On computing time-optimal motions of robotic manipulators in the presence of obstacles." *IEEE Trans. on Robotics and Automation* **7** (Dec 1991).

[61] S. LaValle and J. Kuffner. "Randomized kinodynamic planning." In *IEEE Int. Conf. Robot. & Autom.* (1999).

[62] P. Cheng and S. LaValle. "Reducing metric sensitivity in randomized trajectory design." In *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems* (2001).

[63] H. Niederreiter. *Random Number Generation and Quasi-Monte-Carlo Methods* (Society for Industrial and Applied Mathematics, Philadelphia, USA, 1992).

[64] G. Russell and R. Miles. *Volumetric visualization of 3D data* (Taylor & Francis, Bristol, PA, 1995).

[65] X. Décoret, F. Durand, F. Sillion, and J. Dorsey. "Billboard clouds for extreme model simplification." *ACM Transactions on Graphics* **22** 689–696 (2003).

[66] D. Bertsekas. "Convergence in discretization procedures in dynamic programming." *IEEE Trans. Autom. Control* **20** 415–419 (June 1975).

[67] J.-P. Laumond, S. Sekhavat, and F. Lamiraux. "Guidelines in nonholonomic motion planning for mobile robots." In *Robot Motion Plannning and Control*, J.-P. Laumond, editor, pp. 1–53 (Springer-Verlag, Berlin, 1998).

[68] D. S. S. Pancanti, L. Pallottino and A. Bicchi. "Motion planning through symbols and lattices." In *IEEE Int. Conf. Robot. & Autom.* (2004).

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

[70] P. Choudhury, B. Stephens, and K. Lynch. "Inverse kinematics-based motion planning for underactuated systems." In *IEEE Int. Conf. Robot. & Autom.* (2004).

[71] P. Cheng. *Reducing RRT Metric Sensitivity for Motion Planning with Differential Constraints*. Master's thesis, Iowa State University, Ames, IA (2001).

[72] K. Goldberg. "Completeness in robot motion planning." In *Proc. 1st Workshop on Algorithmic Foundations of Robotics* (A.K. Peters, Wellesley, MA, 1994).

[73] H. Khalil. *Nonlinear systems* (Nacmillan Pulishing, New York, NY, 2002).

[74] A. Marigo, B. Piccoli, and A. Bicchi. "Reachability analysis for a class of quantized control systems." In *Proc. IEEE Conf. on Decision and Control* (2000).

[75] E. Frazzoli. *Robust Hybrid Control for Autonomous Vehicle Motion Planning*. Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA (June 2001).

[76] P. Rouchon, M. Fliess, M. Levine, and P. Martin. "Flatness, motion planning, and trailer systems." In *Proc. IEEE Conf. on Decsion and Control*, pp. 2700–2705 (1993).

[77] P. Cheng, E. Frazzoli, and S. LaValle. "Exploiting group symmetries to improve precision in kinodynamic and nonholonomic planning." In *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems* (2003).

[78] F. Lamiraux, E. Ferre, and E. Vallee. "Kinodynamic motion planning: Connecting exploration trees using trajectory optimization methods." In *IEEE Int. Conf. Robot. & Autom.*, pp. 3987–3992 (2004).

[79] F. Lamiraux, D. Bonnafous, and O. Lefebvre. "Reactive path deformation for nonholonomic mobile robots." *IEEE Trans. on Robot.* **20** 967–977 (December 2004).

[80] S. Kobayashi and K. Nomizu. *Foundations of Differential Geometry. Vol. I*, volume 15 of *Interscience Tracts in Pure and Applied Mathematics* (Interscience Publishers, New York, NY, 1963).

[81] S. Martinez, J. Cortes, and F. Bullo. "A catalog of inverse-kinematics planners for underactuated systems on matrix lie groups." In *IROS*, pp. 625–630 (2003).

[82] J. R. Shewchuk. "An introduction to the conjugate gradient method without the agonizing pain." Available from "http://www-2.cs.cmu.edu/~ jrs/jrspapers.html" (1994).

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

[84] 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).

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

[86] H. Kaindl and G. Kainz. "Bidirectional heuristic search reconsidered." *JAIR* pp. 283–317 (December 1997).

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

[88] P. Krishnaprasad and D. Tsakiris. "Oscillations, se(2)-snakes and motion control: a study of the roller racer." Technical report, University of Maryland (July 1998).

Vita

Peng Cheng

RESEARCH INTERESTS

Robotics, Motion Planning, Computational Geometry, Geometric Control, and Computer Animation

EDUCATION

Ph. D., Computer Science, University of Illinois, Urbana, IL, 2005

M.S., Computer Science, Iowa State University, Ames, IA, 2001

M.E., Electrical Engineering, Tsinghua University, Beijing, China, 1999

B.E., Electrical Engineering, Tsinghua University, Beijing, China, 1996

ACADEMIC EXPERIENCES

Research Assistant, Computer Science Dept., University of Illinois (2001-2005)

Research Assistant, Computer Science Dept., Iowa State University (1999-2001)

Teaching Assistant, Motion Strategy, Computer Science Dept., Iowa State University (Spring 2001)

Teaching Assistant, Computational Geometry, Computer Science Dept. Iowa State University (Fall 2000)

Research Assistant, Electrical Engineering Dept., Tsinghua University (1996-1999)

Reviewer: *Mechatronics*, *American Control Conference*, *IEEE Trans. on Robotics and Automation*, *IEEE Int'l Conference on Robotics and Automation*, *WAFR*, *Robotics: Science and Systems*, *IEEE Conference on Decision and Control*, *European Control Conference*

PUBLICATIONS

1. P. Cheng, Z. Shen, S. M. LaValle, "Using Randomization to Find and Optimize Feasible Trajectories for Nonlinear Systems," in **Proc. Annual Allerton Conf. on Communications, Control, and Computing**, 2000.

2. P. Cheng and S. M. LaValle,"Reducing Metric Sensitivity in Randomized Trajectory Design," in **Proc. IEEE/RSJ/GI Int'l Conf. on Intelligent Robots and Systems**, 2001.

3. P. Cheng, Z. Shen, S. M. LaValle, "RRT-Based Trajectory Design for Autonomous Automobiles and Spacecraft," in **Archives of Control Sciences, Vol. 11(XLVII), No. 3-4, pages 167-194, 2001.**

4. P. Cheng, "Reducing RRT metric sensitivity for motion planning with differential constraints," Masters thesis, Peng Cheng, Iowa State University, 2001.

5. P. Cheng and S. M. LaValle, "Resolution Complete Rapidly-Exploring Random Trees," in **Proc. IEEE Int'l Conf. on Robotics and Automation**, 2002.

6. P. Cheng, E. Frazzoli, S. M. LaValle, "Exploiting Group Symmetries to Improve Precision in Kinodynamic and Nonholonomic Planning," in **IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems**, 2003.

7. P. Cheng, E. Frazzoli, S. M. LaValle, "Improving the Performance of Sampling-Based Planners by Using a Symmetry-Exploiting Gap Reduction Algorithm," in **Proc. IEEE Int'l Conf. on Robotics and Automation**, 2004.

8. P. Cheng, S. M. LaValle, "Resolution completeness for sampling-based motion planning with differential constraints," submitted to **International Journal of Robotics Research**.

9. S. Chitta, P. Cheng, E. Frazzoli, V. Kumar, "RoboTrikke, A Novel Undulatory Locomotion System," in **Proc. IEEE Int'l Conf. on Robotics and Automation**, 2005.

10. S. Lindemann, P. Cheng, "Iteratively Locating Voronoi Vertices for Dispersion Estimation," in **Proc. IEEE Int'l Conf. on Robotics and Automation**, 2005.

11. J. O'Kane, B. Tovar, P. Cheng, S. M. LaValle, "Algorithms for Planning under Uncertainty in Prediction and Sensing," in **Autonomous Mobile Robots: Sensing, Control, Decision-Making, and Applications**, S.S. Ge and F. L. Lewis (Eds.), Series in Control Engineering, Marcel Dekker, Later 2005 or earlier 2006.