Efficient Mobile Robot Path Planning by Voronoi-based Heuristic

Von der Fakultät für Elektrotechnik und Informatik
der Gottfried Wilhelm Leibniz Universität Hannover
zur Erlangung des akademischen Grades

Doktor-Ingenieur
(Dr.-Ing.)

genehmigte Dissertation

von M. Sc. Qi Wang
geboren am 20.01.1982
in Shanxi, China

2015

Referent:	Prof. Dr.-Ing. Bernardo Wagner
Korreferent:	Prof. Dr.-Ing. Christian Müller-Schloer

Tag der Promotion: 1. Dezember 2015

Kurzfassung

Wegen nichtholonomer Einschränkungen ist die Bahnplanung für autoähnliche Roboter eine anspruchsvolle Aufgabe. Der wichtigste Beitrag dieser Dissertation ist es zu ermitteln, wie man mit nichtholonomen Einschränkungen und dem lokalen Minimum der Bahnplanung autoähnlicher mobiler Roboter umgehen kann. Zudem soll dessen Echtzeitleistung verbessert werden. Eine neue Voronoi-basierte Heuristik wird vorgeschlagen, welche die Suche nach dem Ziel viel schneller durchführen kann. Die Komplexität und die Rechenzeit der Bahnplanung autoähnlicher Roboter werden durch die Voronoi-basierte Heuristik erheblich reduziert.

Zuerst wird ein neuer primitiver Trajektoriensatz anhand der nichtholonomen Einschränkungen definiert. Dieser kann mehrfach verwendet werden, um einen Suchbaum für die Untersuchung des Suchraums zu konstruieren; somit ist der erzeugte Pfad immer manövrierbar für autoähnliche Roboter. Der definierte primitive Trajektoriensatz ist sehr einfach, aber dennoch können mit unterschiedlichen Kombinationen verschiedene Manöver erzeugt werden, die Einfachheit und Vielfältigkeit enthalten. Später wird ein neuer Ansatz vorgeschlagen, um die Veränderung der Krümmung entlang der Bahn zu reduzieren, so dass die Manöver mit aggressiver Lenkung weitgehend eliminiert werden.

Angesichts der nichtholonomen Einschränkungen muss der Lenkwinkel und die Richtung in die Bahnplanung integriert werden, um einen durchführbaren Weg zu produzieren, dem auch autoähnliche Roboter physikalisch folgen können. Dies kann die Komplexität der Bahnplanung erheblich erhöhen und fast unendlich Rechenzeit und Speicherplatz kosten, besonders unter großen Clusterumgebungen. Die Voronoi basierte Heuristik wird vorgeschlagen, um dieses Problem zu umgehen. Das generalisierte Voronoi Diagramm ist eine hohe Abstraktion der Umgebung, die immer als ideale Roadmap gilt. Mithilfe der Voronoi-basierten Heuristik ist die Suche in der Lage leicht durch den Raum zu navigieren, alle Arten von lokalen Minima zu vermeiden und somit die Rechenzeit des Bahnplanungsprozesses weitgehend zu reduzieren.

Globale und lokale Bahnplanungsprozesse haben ihre eigenen Stärken und Schwächen. Globale Bahnplanung wird bevorzugt für eine statische globale Umgebung angewendet, während lokale Bahnplanung mehr für eine lokale dynamische Umgebung geeignet ist. Um während des Folgens eines globalen Pfads zu einem Ziel mit dynamischen Hindernissen umzugehen, wird ein neuer Ansatz vorgeschlagen. Dabei werden beide Prozesse integriert, um ihre jeweilige Schwäche zu überwinden.

Um die vorgeschlagenen Ansätze im Rahmen dieser Arbeit zu überprüfen, werden die Algorithmen in einer simulierten Umgebung getestet. Da die Systemdynamik des autoähnlichen Roboters ebenfalls eine wichtige Rolle spielt, was die Leistung des Bahnplaners beeinflussen kann, wird ein autoähnlicher Roboter modelliert, um die Prüfung der vorgeschlagenen Ansätze zu erleichtern.

Schlagworte: autoähnliche mobile Roboter, Bahnplanung, Voronoi

Abstract

Nonholonomic constraints based path planning for car-like robots is a challenging task. The most important contribution of this dissertation is how to deal with nonholonomic constraints and the local minimum of car-like mobile robot path planning, as well as improvements of its real-time performance. A new Voronoi-based heuristic, which is able to guide the search to the goal much faster, is proposed. Voronoi-based heuristic significantly reduces the complexity and the computation time of car-like robot path planning.

First of all, a novel primitive trajectory set is defined based on the nonholonomic constraints. It can be applied repeatedly to construct a search tree for space exploration; the produced path is, thus, always maneuverable for car-like robots. The defined primitive trajectory set is very simple, but with different trajectory combinations, various maneuvers that maintain both simplicity and diversity can nevertheless be produced. Later, a new approach is proposed to reduce the variation of the curvature along the path during the search, so that the maneuvers of aggressive steering can be largely eliminated.

Given the nonholonomic constraints, the steering angle and orientation must also be integrated in path planning to produce a feasible path that the car-like robot can physically follow. This highly promotes the complexity of path planning and can cost almost infinite computation time and memory space, especially under large clustered environments. The Voronoi-based heuristic is proposed to deal with this problem. The generalized Voronoi diagram is a high abstraction of the environment which is always regarded as the ideal roadmap. With the help of the Voronoi-based heuristic, the search is able to easily navigate through the space and avoid all kinds of local minima. Thus, the computation time of the path planning process is largely reduced.

Both global and local path-planning processes have their own strengths and weaknesses. Global path planning is preferable for a static global environment whereas local path planning is more suitable for a local dynamic environment. In order to cope with dynamic obstacles while following the global path to a goal, this work proposes a new approach to integrate both processes to counteract their shortcomings.

To evaluate the proposed approaches within this work, the algorithms are tested under a simulated environment. Since the system dynamics of the car-like robot also play an important role that may affect the performance of the path planner, an intricate car-like robot is modeled to facilitate the testing of the proposed approaches.

key words: car-like mobile robot, path planning, Voronoi

Acknowledgments

First and foremost I would like to thank my adviser, Professor Bernardo Wagner, who invited me to Germany and gave me a valuable chance to work in his group, I learned a lot there about robotic technologies. He was always patient enough to discuss about my work with me and helped me efficiently schedule my time, so I could focus on my work without any distraction. This dissertation would not have been possible without his kind words of caution and advice.

I would like to thank my colleagues Marco Langerwisch, Frauke Wübbold, Christian Wieghardt, Matthias Hentschel, Paul Fritsche and Markus Wulfmeier for always being there for me, and helping me correct the errors in my work and giving me valuable support and advice that largely improved the quality of my dissertation. I would also like to thank Jorge Nieto-Madrid for always taking care of me with his kindness and helping me get acquainted with the environment which made my life in Germany much easier.

I thank Mechthild Wilke, Wolfgang Möller, Andreas Weiner and Jörg Adolf for their kind help with frequent office affairs.

I also want to thank Professor Gong Jianwei, and Yu Zhang from the Intelligent Vehicle Research Center, Beijing Institute of Technology.

Finally, I would like to thank my parents for their constant support and encouragement throughout my life.

Contents

List of Abbreviations and Symbols

Definition of Primitive Trajectory

S^3	3D continuous configuration space
S^3_d	3D discrete configuration space
θ_i	Discrete orientation of robot
α	Steering angle
α_{max}	Maximum steering angle
ω	Steering rate
ω_{max}	Maximum steering rate
θ_{min}	Minimum orientation step
L	Wheelbase of car-like robot
W	Wheel gauge of car-like robot
M_{front}	Front margin of car-like robot
M_{rear}	Rear margin of car-like robot
M_{side}	Side margin of car-like robot
T_α	Primitive trajectory with steering angle α
T_α^{foot}	Footprint trajectory of T_α

Evaluation of Primitive Trajectory

Δt_{steer}	Time cost with maximum steering rate along primitive trajectory
Δt_{clear}	Minimum time cost affected by clearance along primitive trajectory
Δt_{min}	Time cost with maximum linear velocity along primitive trajectory
Δt	Estimated time cost of moving along primitive trajectory
κ_{steer}	Steering rate coefficient
κ_{clear}	Clearance coefficient

Voronoi-based Thinning

GVD	Generalized voronoi diagram
CPT	Classic parallel thinning

VPT	Voronoi-based parallel thinning
Υ_{top}	Thinning pattern set from top-left side
Υ_{bottom}	Thinning pattern set from right-bottom side
$clear[]$	Clearance map

Global Path Planning

ζ_i	Discrete configuration (x_i, y_i, θ_i) in S_d^3
ζ_{start}	Start configuration
ζ_{goal}	Goal configuration
c_i	Position (x_i, y_i) of ζ_i
n_i	Node i in exploring tree
n_{start}	Startnode on ζ_{start}
n_{goal}	Goal node on ζ_{goal}
$g(n_i)$	Path cost of n_i
$h(n_i)$	Heuristic cost of n_i
$f(n_i)$	Estimated cost of n_i
X_{GVD}	Set of GVD positions
X_{goal}	Goal zone
χ_i	GVD position of ζ_i
χ_{start}	GVD position of ζ_{start}
χ_{goal}	GVD position of ζ_{goal}
$cost[]$	Cost distribution over GVD
v_{max}	Maximum linear velocity during cost distribution
v_{min}	Minimum linear velocity during cost distribution
τ_{clear}	Clearance coefficient during cost distribution
NLM	Nonholonomic local minimum
χ_{nlm}	GVD position of nonholonomic local minimum

Local Path Planning

$path\_clear[]$	Corridor clearance map
P_{global}	Global path
P_{norm}	Normative path
$L_{GVD}(n_{start})$	Connection from start to GVD
$L_{GVD}(n_{goal})$	Connection from goal to GVD
X_{path}	GVD section of normative path P_{norm}
p_i	Path position on P_{norm}
p_{robot}	Current path position of robot on P_{norm}
$norm\_cost[]$	Cost distribution along normative path

Ω_n	Corridor clearance threshold
Φ_{clock}	Timeout clock
Ω_t	Timeout threshold

List of Figures

List of Algorithms

List of Tables

Part I

Background

Chapter 1

Introduction

In modern times, robotic technologies influence almost every aspect of human life, but the mobility of a nonholonomic mobile robot remains a challenge. This dissertation deals with some fundamental path planning problems of the nonholonomic mobile robot under large-scale, clustered, planar environments, particularly for car-like robots.

1.1 Motivation

Generally speaking, there are three types of wheeled mobile robots: omnidirectional robots; differential wheeled robots; and car-like robots. The omnidirectional robot [Fig. 1.1(a)] provides the most flexible mobility and requires almost no constraints of maneuvers; it largely simplifies the process of path planning. However, it is only applicable to clean, indoor environments with a very slow speed due to the high complexity of the mechanical structure of omnidirectional wheels.

(a) (b) (c)

Figure 1.1: a) Omnidirectional robot; b) Differential wheeled robot; c) Car-like robot

Differential wheeled [Fig. 1.1(b)] and car-like mobile robots [1.1(c)] are nonholonomic mobile robots. The steering systems of such robots are quite simple and, therefore, they

can move faster and run on rough and dirty surfaces. The shortcoming of the differential wheeled robot is its distinctive lateral movement of wheels while cornering, which produces large friction between the wheels and the ground. Thus, the robot may lose a large amount of energy which can affect the linear velocity. Therefore, differential wheeled robots are still not applicable to high-speed movement in large environments.

Compared to omnidirectional and differential wheeled mobile robots, the car-like robot requires a simple mechanical structure and less energy cost, thus providing the most preferable mobility under large-scale environments. However, the nonholonomic constraints of the car-like robot pose a challenge to path planning; for example, even a simple parallel parking may become very complex since it involves several frequent steering moves with forward and backward maneuvers. Despite the numerous researches on car-like robots in the last few years [4–7], the existing solutions for such robots still mostly concentrate on an on-road environment, where the nonholonomic constraints are guaranteed by the road itself and, therefore, can be ignored or simplified during path planning. However, in off-road, planar environments with more sophisticated and no predefined road information, the noholonomic constraints must be guaranteed by the planner. As will be mentioned in the next chapter, because the nonholonomic constraints largely increase the complexity of path planning, existing solutions are only applicable to simple and small spaces. In planar, large and clustered environments, especially with the existence of a large number of concave local minima, the search could be trapped. The search either takes a very long computation time or produces a poor path, or even fails. There are still some fundamental problems concerning car-like robot path planning and, therefore, a more intelligent path planning algorithm is required. This is the motivation of this work.

This work concentrates on service mobile robot path planning based on car-like platforms under the planar human environment, such as shopping center, airport, residential area, warehouse, factory, etc., where the ground surface is assumed to be flat. The roughness, slop, or unevenness of the ground are therefore not considered, since those factors are more generally concerning the field robot which is a different topic that will not be covered in this work.

1.2 Problem Statement

One of the problems regarding search-based path planning is the local minimum that is caused by the inaccuracy of heuristic. Heuristic helps the search become more goal-oriented and reach the goal faster. However, the inaccuracy of heuristic can also lead the search to the wrong direction and make it trapped, thereby necessitating more computation cost. The path planning of omnidirectional or differential wheeled robots can be performed in a 2D space and therefore requires relatively low computation cost, where the effect of the local minimum can be mostly neglected. However, the nonholonomic constraints of the car-like robot greatly add to the complexity of path planning. As will be shown in Section 2.4.1, the path planning of the car-like robot needs to be performed in a 3D or even higher dimensional

space. Moreover, the factors causing local minimum for car-like robot are more complex. It can be caused by obstacle constraints and nonholonomic constraints, separately or combined. The local minimum can happen during the process of global or local path planning. Hence, the problem of the local minimum for car-like robots is much more dominant than for omnidirectional or differential wheeled robots. The local minimum problem for car-like robots is decomposed into several fundamental problems pertaining to nonholonomic constraints, global path planning, and local path planning. The present dissertation seeks to solve these problems.

Nonholonomic Constraints

Nonholonomic constraints pose a challenge for car-like robot path planning that not only requires the continuity of the curvature along the produced path, but also stipulates no violation of the minimum turning radius that the robot can perform. Owing to nonholonomic constraints, the local minimum problem for car-like robots becomes more complex, thus increasing the computation cost of path planning.

Primitive Trajectory Set

In recent years, the motion primitives-based path planning has attracted a great amount of interest as it provides a successful tool to comply with the nonholonomic constraints. In terms of the mobile robot, the motion primitives are a set of primitive trajectories by sampling the control space. Therefore, the primitive trajectories take care of the nonholonomic constraints, so the planner can solely focus on the obstacle constraints. However, it is always difficult to define a primitive trajectory set, which satisfies the system constraints without reducing the diversity of the control space. Some of the existing approaches tend to make a large trajectory set by discretizing the control space as fine as possible [8]. This is very difficult to implement and requires a large amount of work to be done in advance. Additionally, it can also increase the time cost of the search being trapped by the local minimum, since more primitive trajectories need to be tested in the process of path planning. Other approaches prefer to make a simple trajectory set [9], but may lose the diversity of the control space and, thus, reduce the optimality of the produced path. This leads to the problem of how to define a primitive trajectory set in a simple form while simultaneously maintaining the diversity of the control space.

Aggressive Steering

The most particular feature that differentiates a car-like robot from other types of robots is the Ackermann steering system [10]. As will be discussed in Section 3, the turning radius of the car-like robot depends directly on the steering angle. Since the steering angle is supposed to vary continuously, the curvature of the trajectory must also be continuous along the trajectory, failing which the robot has to frequently stop and perform a steering action

at the curvature-discontinuous points. To find a feasible path with a continuous curvature is no longer difficult with many solutions proposed. These will be discussed in Section 2.4.1. However, few of them have ever efficiently minimized the variation of the curvature along the path. A large variation in the curvature will cause aggressive steering, i.e. to perform a large-scale steering angle in a very short time. The aggressive steering will result in a large torque on the steering wheel. This will not only require a high-energy cost but may also damage the steering system. Therefore, having to avoid aggressive steering by minimizing the variation of the curvature along the path also becomes a problem.

Local Minimum of Search-based Path Planning

Motion primitives are utilized to explore the space and produce a path that meets the nonholonomic constraints. In order to avoid uniformly exploring the entire space, which may require a large computation time and memory space, the search-based algorithm applies heuristic to guide the search, moving towards the direction of the goal. Intensive researches of search-based path planning have been done in the last few decades, but the topic of the heuristic has barely been touched. The Euclidean distance is still the most widely used heuristic. Although a few works have been done on heuristic [11–21], they all degenerate to the Euclidean distance-based heuristic for long-range path planning. The Euclidean-based heuristic brings a major drawback that the search may be trapped if concave obstacles happen to be located in the middle of the goal and the starting point. This is known as the local minimum problem in path planning. In a clustered environment, such local minimum could be everywhere and will dramatically slow down the search.

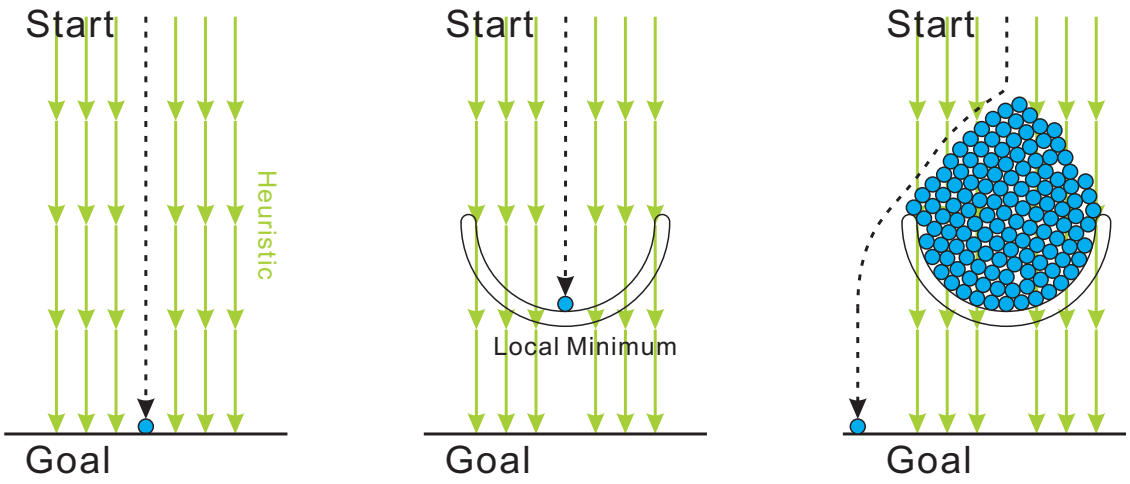


Figure 1.2: Path planning with Euclidean-based heuristic: a) Without local minimum; b) With local minimum; c) Goal can only be reached when local minimum is filled

The problem of the local minimum can be illustrated as Fig. 1.2. The small blue ball is initiated from the start and is supposed to reach the goal. The Euclidean-based heuristic in this case can be regarded as the gravity field (shown as light green arrows) that pulls the blue ball moving towards the goal. The blue ball may land directly on the goal if nothing exists in between [as in Fig. 1.2(a)]. However, if there is an obstacle on the way [as in Fig. 1.2(b)], the blue ball may get trapped and then the obstacle would form a local minimum. As shown in Fig. 1.2(c), the current solution for the local minimum is simply to drop more blue balls in the local minimum until it is full, and then the next blue ball can eventually reach the goal. Unfortunately, filling the local minimum takes too long, especially in a large clustered environment. This is the major problem that this work seeks to solve.

Roadmap

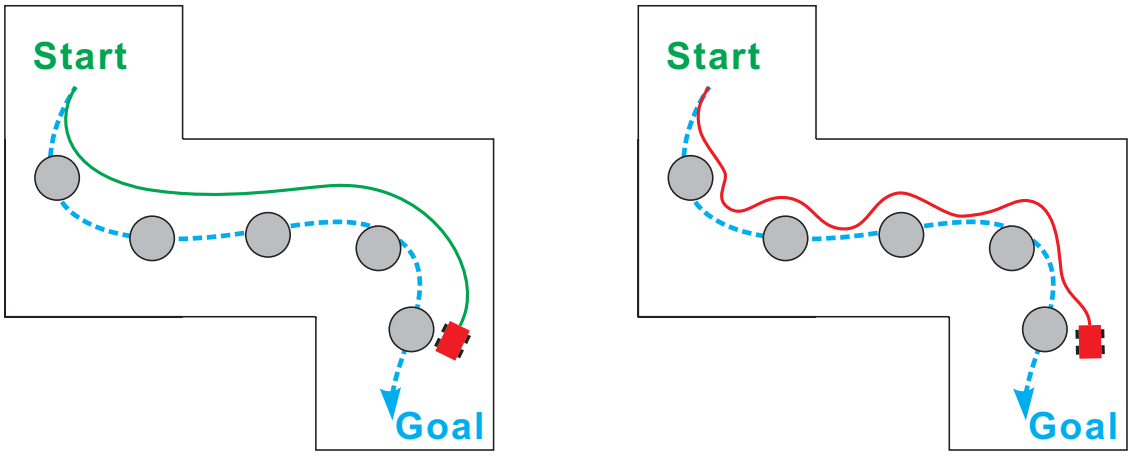
The roadmap is an abstract description of the environment, which connects the entire free space with a graph structure. Since the graph only consists of a very limited number of nodes, the path planning problem can be greatly simplified by just searching a sequence of connected nodes along the graph that connects both the start and the goal instead of the whole space. The computation of the roadmap-based search is very fast and, therefore, can hardly be affected by the local minimum. In this dissertation, a roadmap is used as a guidance that leads the nonholonomic path planning away from the local minimum and largely reduces the computation cost.

Unfortunately, there are two problems of roadmap-based path planning. First, it is always hard to create a roadmap that describes the environment in an optimal way. As will be discussed later in Section 2.1.2, the computation cost of such a roadmap is very high which makes such an approach very sensitive to the dynamics of the environment. Second, the resulting path by searching the roadmap can hardly guarantee the nonholonomic constraints. Since the path can pass through some very winding and narrow corridors that the nonholonomic robot cannot follow, there is a problem of how to apply the roadmap to serve nonholonomic path planning. These two problems are also supposed to be solved in this dissertation.

Integration of Global and Local Path Planning

Global path planning is supposed to produce a global path based on a premade static map, which can be guaranteed to be collision-free only against static obstacles on the map. However, the global path can be locked when dynamic obstacles exist and can, therefore, not be applied directly. In such cases, reproducing the whole global path may take too long to catch up with the dynamics in a large environment. The existing solutions are making local changes along the global path to avoid dynamic obstacles. However, the changed path cannot vary too much since obstacle avoidance always has to be done based on the original global path. This will tremendously reduce the flexibility of the local planner and increase

the risk of it being trapped by the local minimum. Thus, the local planner can fail sometimes even when a feasible path does exist.



(a) (b)

Figure 1.3: Dashed blue spline is global path: a) Green spline is ideal local path; b) Red spline is actual local path.

Even when the local planner has successfully avoided the obstacles, the varied local path is no longer optimal. As in Fig. 1.3(a), the dashed blue spline represents the precalculated global path and there are several dynamic obstacles laid on the global path (shown as circular objects). The green spline represents the ideal local path. Since the robot always tries to follow the global path, it will have to leave the global path for a while if there is an obstacle laid in the way and move back to the global path after it has bypassed the obstacle, i.e., simultaneous path following and obstacle avoidance (SPFOA) [22, 23]. Imaginably, when there are multiple obstacles laid on the original global path [as in Fig. 1.3(b)], the robot has to frequently leave and return to the global path, which eventually generates a very snaky local path [shown as the red spline in Fig. 1.3(b)]. In this case, the global path serves as not only a guidance that leads the robot to the goal, but also an extra constraint which requires the robot to always follow. As a result, the optimality of the local path greatly deteriorates. Therefore, effectively integrating the local and global path planners, while maintaining both optimality and flexibility of the path, becomes a challenge.

1.3 Concept of Solutions

This work proposes solutions to the local minimum problem of both global path planning and local path planning. The search-based path planning is carried out by repeatedly applying a new primitive trajectory set that defines the basic behavior of the nonholonomic mobile robot. The continuity of the curvature along the trajectory is achieved by smoothing the

primitive path with the Bézier spline [24]. Besides, the aggressive steering is also avoided by attaching large costs to the trajectories of aggressive steering so that the trajectories of peaceful steering become more preferable during the search. Therefore, the variation of the curvature along the path can be minimized directly during the search.

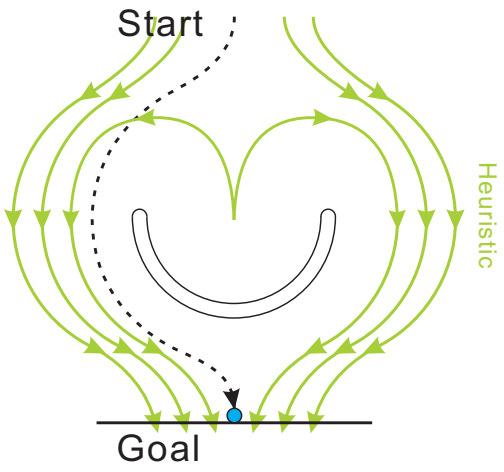


Figure 1.4: Heuristic navigates its way around concave obstacle

As mentioned in Section 1.2, the local minimum problem in Fig. 1.2 results from Euclidean-based heuristic that pulls the search directly toward the goal. Therefore, a new type of heuristic that reflects the accurate cost to the goal is proposed. The heuristic cost is measured in accordance with a roadmap that is constructed on the basis of a generalized Voronoi diagram (GVD). Generally speaking, the GVD is simply a collection of the medial lines between the obstacles and precisely reflects the geometry of the environment. Therefore, the GVD is always considered as an ideal representation of the environment and the heuristic value measured along the GVD becomes more accurate. As a result, the search will no longer be attracted to the local minimum and the computation time of the search can be greatly reduced. As shown in Fig. 1.4, the proposed Voronoi-based heuristic can also be illustrated as a gravity field. The difference is that the heuristic (shown as arrows along the light green lines) now navigates its way around the concave obstacle, so that the blue object can reach the goal directly without being trapped in the local minimum.

An improved thinning algorithm is also proposed which not only simplifies the construction of the GVD-based roadmap but also provides the connections between the free space and the GVD. Therefore, any position in the free space can be easily evaluated based on the connection to the GVD. This makes the computation of the Voronoi-based heuristic as fast as that of the Euclidean-based heuristic while largely promoting the accuracy of the heuristic at the same time.

To effectively integrate global and local path planning, the global path will not be used directly but will be transformed into a sequence of corridor sections that the global path

passes through. The corridor sections define the maximum space along the global path. The robot is not required to always follow the global path but only limited inside the corridors. This clearly endues the robot more flexibility to avoid dynamic obstacles and reduces the risk of being trapped by the local minimum. Meanwhile, the path corridor-based heuristic is also extracted from the global path to attract and navigate local path planning efficiently through the corridors.

1.4 Structure of Chapters

Fig. 1.5 shows the structure of this dissertation. The chained yellow boxes represent the major components of the path-planning algorithm. Each component takes the input from the previous component (shown as the pink boxes). The green boxes show the input from the environment and the blue box is the mobile robot platform that takes velocity v and steering angle α as inputs. Basically speaking, this dissertation consists of four parts.

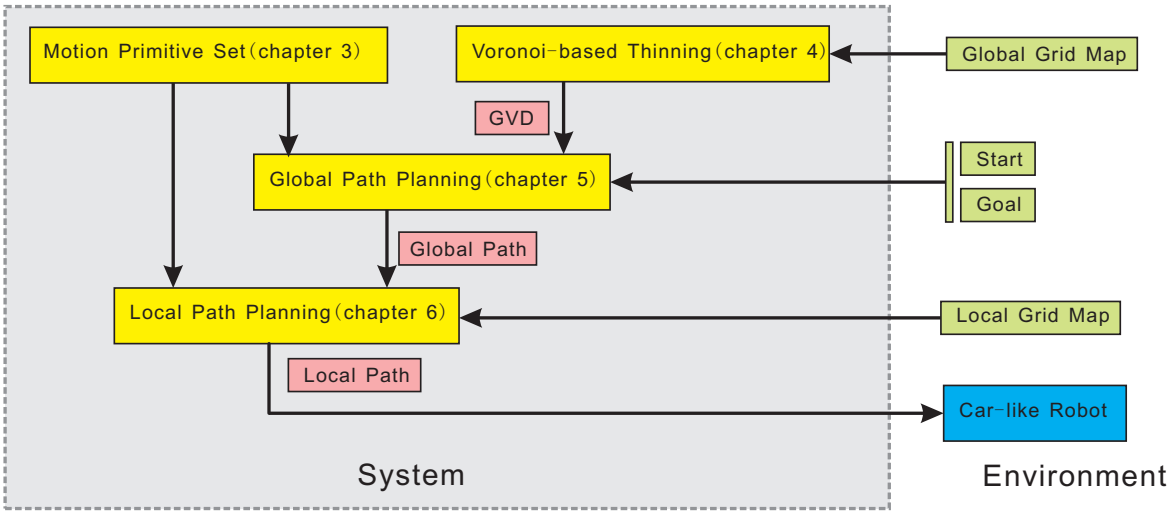


Figure 1.5: New Voronoi-based path planning process
(Details of proposed approach are described in more detail in indicated chapters).

Chapter 2 in **Part I** introduces the current state of nonholonomic mobile robot path planning as well as the problems that are supposed to be solved in this dissertation.

Part II (**Chapter 3**) proposes a new primitive trajectory set to solve the problem of the nonholonomic constraints for the car-like robot path planning. With various combinations of primitive trajectories, all kinds of maneuvers can be produced.

Part III proposes several Voronoi-based algorithms to solve global, local as well as multi robot path planning problems that are separated into four chapters. **Chapter 4** proposes a new thinning algorithm that provides a very simple way to extract the GVD

based roadmap from the grid map and serves as a tool to facilitate path planning algorithms in other chapters. **Chapter 5** concerns global path planning. Primitive trajectory set-based path planning is applied with the help of the extracted GVD roadmap that provides a more accurate heuristic estimation of the goal. Compared to the traditional Euclidean-based heuristic, the Voronoi-based heuristic largely reduces the computation time of the search. A new local path planning algorithm is proposed in **Chapter 6**. Instead of following a specific global path, the local planner produces a path corridor which defines the maximum space along the global path, and the actual local path is produced on the fly within the path corridor. This endues the robot with great flexibility to avoid dynamic obstacles.

Part IV provides the tests and evaluations of the proposed approaches. The approaches are tested on a simulated vehicle illustrated in **Appendix A**. Some experiments are made in **Chapter 7** to demonstrate the efficiency and usefulness of the proposed solutions. It shows the proposed solutions only require very low computation time and memory cost without reducing the optimality of the produced path.

Chapter 2

State of the Art in Path Planning

In this chapter, the current state of different aspects concerning nonholonomic mobile robot path planning is introduced. Path planning must be applied to a certain type of map representation. The actual map representation largely affects the behavior of path planning. Several popular map representations are presented and evaluated followed by a discussion on three major types of nonholonomic path planning algorithms which brings out the importance of heuristic for path planning. The focus then shifts to the current state of heuristic as well as their problems, followed by motion primitives-based path planning with an illustration of how motion primitives can affect the efficiency and flexibility of path planning. As mentioned in Section 1.2, the local planner plays an important role in dealing with dynamic obstacles based on the global path. The integration of local and global planner highly affects the real-time performance of the robot, and the existing local planners and their major shortcomings are also addressed in this chapter.

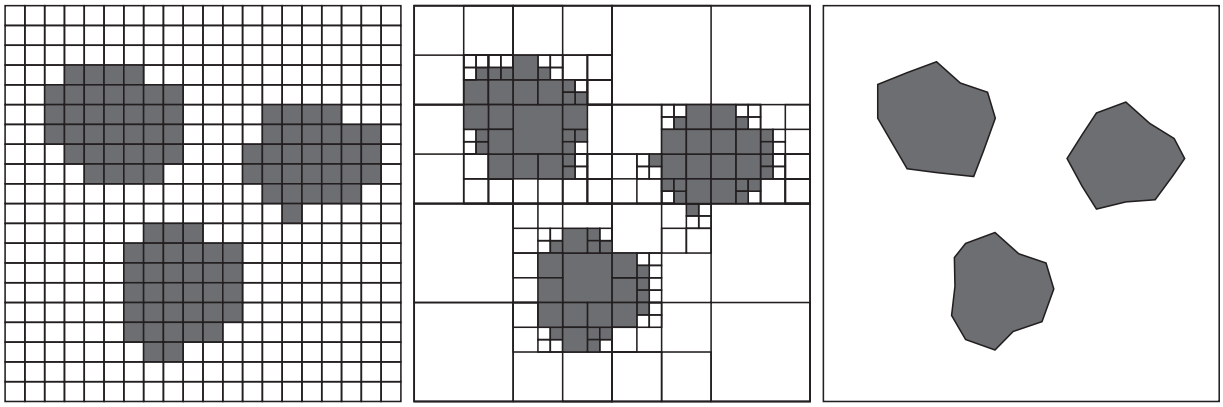
2.1 Map Representations

The map representation is the first step of path planning. A well-defined map structure can dramatically reduce the complexity of path planning. The map should be not only easy to create and modify, but also simple to query. Generally speaking, mobile robot path planning uses two major types of map: the metric maps and topological maps. Both map types have their significant drawbacks and, therefore, current researches on mobile robot path planning generally use a hybrid form [25–28] of metric and topological maps. As mentioned in Section 1.3, the proposed strategy to avoid the local minimum during the path planning process is also based on a hybrid map.

2.1.1 Metric Maps

Metric maps provide an accurate description of the environments that record the precise locations of all the obstacles in space. They are more suitable for precise collision-checking

during the path planning process. Grid maps [29] are the most widely used example of metric maps for mobile robot. Grid maps discretize the space with equally spaced cells [as shown in Fig. 2.1(a)], and each cell can be located with the x-y coordinate and is attached to a value specifying whether or not the cell is occupied. The gray cells represent the obstacle cells. Grid maps, which are very simple to query, provide a simple way for collision checking. However, grid maps are still low-level map types. For wide, clustered environments, the search space can be very large, searching directly over grid map can easily be trapped by local minimum and, thus, requires relatively large computation cost.



(a) (b) (c)

Figure 2.1: a) Grid map; d) Quadtrees; c) Polygonal map

Increasing the resolution of the grid map may result in a higher memory space cost, whereas lower resolution can make the narrow corridors disappear, thereby reducing the map precision. Unlike grid maps, quadtrees provide a solution to adaptively change the resolutions [30–32] of the map, which reduces the memory cost but preserves the precision [as shown in Fig. 2.1(b)]. Quadtrees initially treat the map as a single cell. If the cell consists of both free space and obstacles, then quadtrees equally divide the cell into four subcells. The same process is iteratively applied to the subcells until the minimum size has been reached. However, compared with a regular grid map, quadtrees are very difficult to apply with the path planning algorithm directly [33, 34], since any cell in quadtrees-based map can no longer be easily located based on its coordinates and its neighboring cells can only be returned by recursively search the quadtrees. Although the fundamental advantage of quadtrees is their low memory cost, such an advantage become less evident in the recent years due to the falling cost of the computer memory.

Alternatively, polygonal maps [Fig .2.1(c)] pose another way to represent the obstacles with a sequence of geometric features. The polygonal map is applicable for the environment where obstacles are mostly in a relative regular form, for example an indoor space where walls, tables or chairs can always be represented with regular shapes. Such environments can be represented by polygonal maps at a very small memory cost. However, under more complex

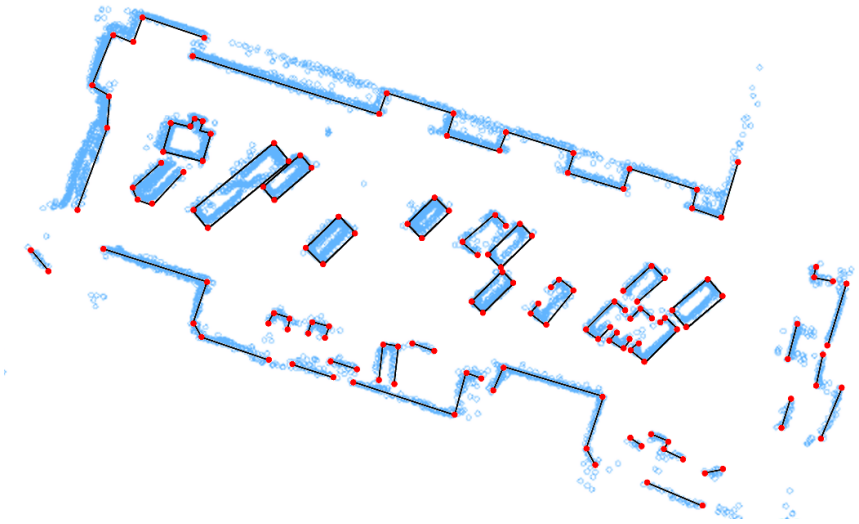


Figure 2.2: Extraction of geometric features from raw data of ranger sensor (image from [1])

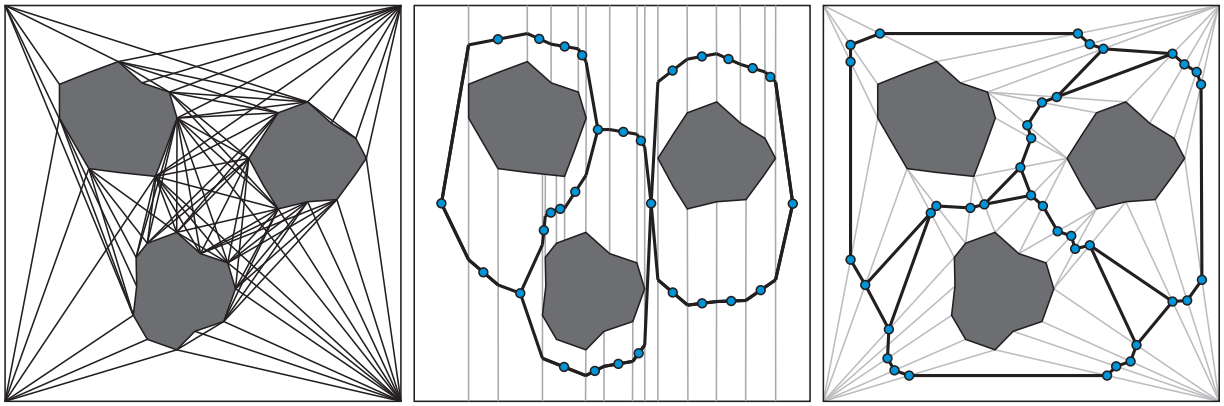
environments with various types of objects, polygonal maps can be very difficult to use. As shown in Fig. 2.2, the blue points represent the raw data gathered with the ranger sensor and the black lines represent the extracted geometric features. It can be seen that the raw data is very noisy. Some raw points cannot be efficiently defined directly with polygonal shapes. The extraction of the geometric features in a real-word environment can be very challenging [35–37]. Therefore, polygonal maps are more useful for navigation problems under simulated environments or computer games, where all objects are geometrically predefined.

As mentioned, the grid map is very simple to create and query, moreover, the falling cost of the computer memory in recent years also eliminates the limitation of grid map representing large scale environments. All of this makes the grid map more effective compared with other types of map. The present dissertation uses the grid map combined with a topological map to serve the path planning of the car-like robot.

2.1.2 Topological Maps

A topological map is a highly extracted abstraction of an environment, and represents the environment with a graph structure that connects all traversable areas. The topological map has the advantage of being able to handle a large environment with a small and simple data structure, largely reducing both memory cost and complexity of path planning. This provides an opportunity to use the topological map as a roadmap to guide the search away from the local minimum (as introduced in Section 1.3). However, as it will be discussed later in this section, accurate and consistent topological maps are considerably difficult to create in clustered, large-scale environments.

Fig. 2.3(a) shows the same map as Fig. 2.1(c). There are black lines that connect all intervisible vertexes of the polygons, i.e. the lines that connect any two vertexes without



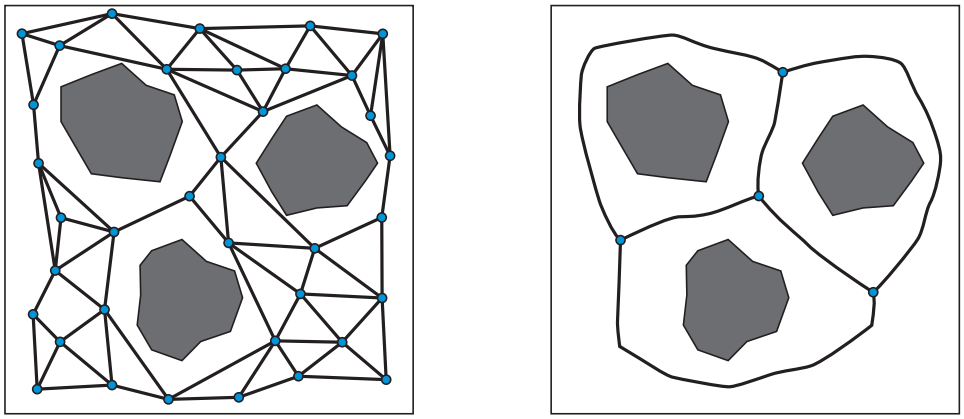
(a) (b) (c)

Figure 2.3: a) Visibility graph; b) Trapezoidal decomposition; d) Triangular decomposition

intersecting with the obstacle. These lines comprise a graph that can be used for path planning as a topological map to find the Euclidean shortest path among a set of polygonal obstacles [38]. The definition of a visibility graph is simple and straightforward. However, the edges of the visibility graph dramatically increase with the number of vertexes. This makes the visibility graph very hard to apply and maintain for large, clustered environments that consist of obstacles with a large number of vertexes.

Polygon decomposition [39–41] divides the space into small fractions, and a topological map can be constructed by connecting the neighboring fractions. As in Fig. 2.3(b), the trapezoidal decomposition divides the space into small trapezoids with the gray vertical lines that connect the vertexes as well as the upper and lower boundaries. The topological map is constructed by connecting the center of the divided neighboring trapezoids. The triangular decomposition [as in Fig. 2.3(c)] divides the space into small pieces of triangles. The back lines connecting the center of the edges of the triangles form a topological map. Both topological maps created with trapezoidal and triangular decompositions are complete; a path can be found as long as it does exist between the start and the goal. However, because of the drawbacks of polygonal maps mentioned in Section 2.1.1, a polygonal decomposition can be hardly applied in a real-word environment.

Fig. 2.4(a) is the topological map created by the probabilistic roadmap method (PRM) [42]; the randomized path planning algorithm will be discussed later in this chapter as well. PRM creates a topological map by randomly sampling the nodes in a free space and making connections between them. As shown in Fig. 2.4(a), the created topological map cannot precisely describe the geometries of the space since all the nodes are randomly sampled. Besides, PRM also has the problem of losing the connectivity of narrow corridors since they fail to get enough samples to maintain the connectivity, especially for clustered environments. This could be solved by promoting the sampling intensity but it also increase the overall number of samples and, consequently, the memory and computation cost.



(a) (b)

Figure 2.4: a) Probabilistic roadmap; b) Generalized Voronoi diagram

Because of mentioned drawbacks, the above approaches of building a topological map are not applicable to large, clustered environments. A method to extract the topological map that can accurately describe the geometry of the environment at a very small computation cost is required.

A Voronoi diagram can be regarded as a collection of medial lines between sites. The first work of Voronoi is presented in the work of [43, 44], then a number of algorithms that extract the Voronoi diagram from point-based sites become available [45–47]. To apply the Voronoi diagram to a real-word environment, the sites are extended to general shapes. This transforms the Voronoi diagram into the generalized Voronoi diagram (GVD). Likewise, the GVD is also a collection of the medial lines between the generalized sites [as in Fig. 2.4(b)]. Compared with the other types of topological maps, the GVD [48] is regarded as the most simple and ideal description of the environment. Therefore, the present dissertation uses the GVD as the topological map to enhance the path planning of the car-like robot.

One of the methods to extract the GVD is based on the polygonal map. The GVD is extracted by analyzing the geometrical relationships between the elementary objects [49–52]. However, because of the drawbacks of polygonal maps (Section 2.1.1), such algorithms are not applicable to real-world environments. Apart from the problem of building the polygonal map, computation of the GVD is very time consuming and difficult to implement when the given environment includes a large number of objects in a polygonal form.

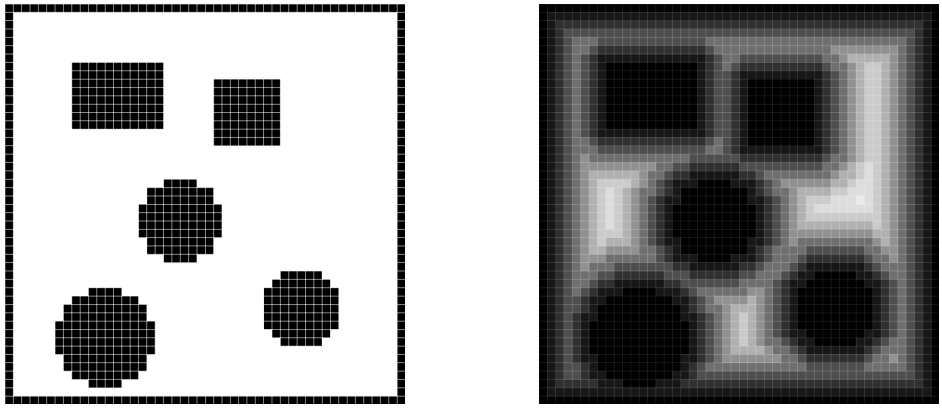
Figure 2.5: Thinning algorithm

Another type of GVD calculation applies the thinning algorithm [53–58]. This algorithm is originally one of the image processing algorithms. It iteratively removes small fractions along the boundary of the object to make the object thinner and thinner until only a very thin line is left (as in Fig. 2.5). The thinning algorithm can be directly applied to a grid map, so the conversion of a polygonal map is no longer required. Since the thinning algorithm deals with the grid map cell by cell, the computation time is proportional to its size of the map and independent of its complexity. [59] shows that the cost of GVD calculation with thinning algorithms is very low and even meets the requirement of real-time applications.

Furthermore, since the GVD is extracted from the grid map, the extracted GVD can be naturally mapped to the original grid map. This allows every cell in the grid map to be easily connected to the GVD (as will be introduced later in Chapter 5). The connection between the GVD and the grid map becomes the key to using the GVD to guide the search exploring the grid map.

However, the thinning-based GVD calculation also has its drawbacks. The result of the thinning algorithm does not represent the real GVD. The layout of the grid map gives rise to square effect during the thinning process,making the resulting GVD inaccurate. A Voronoi-based parallel thinning algorithm to solve this problem which precisely extracts the GVD from the grid map at a small time cost will be proposed in Chapter 4.

2.1.3 Clearance Map



(a) (b)

Figure 2.6: a) Grid map; b) Clearance map

Clearance maps also known as distance maps can be produced by applying a distance transformation [60–63] on grid maps. On a clearance map, each cell is assigned a value representing the distance to its nearest obstacle. Fig. 2.6(a) is the grid map of the environment and Fig. 2.6(b) is the produced clearance map based on the grid map. Each cell c_i in the grid map has a corresponding clearance value at the same location on the clearance

map which represents the distance from c_i to its nearest obstacle. Since the clearance map provides an easy way to measure the distance to the obstacles, it becomes a simple tool of collision checking for mobile robot path planning, i.e. the robot may have a collision at a certain point when the clearance value is less than the safety threshold. In Chapter 3, the clearance map is used for path optimization so that the planner may choose the areas with higher clearance or, in other words, the area away from the obstacles. This increases the safety assurance for the robot against the obstacles. In Chapter 4, the clearance map is also used to improve the accuracy of GVD calculation.

2.2 Path Planning Algorithms

The research over nonholonomic robot path planning has been continuing for a few decades and despite great improvements most solutions are still only applicable to small and simple environments. Generally speaking, there are three most intensively discussed types of path planning algorithms concerning nonholonomic mobile robots—the rapidly exploring random tree (RRT), the probabilistic roadmap method (PRM), and search-based path planning. The RRT and the PRM are regarded as randomized path planners, whereas search-based path planning is the deterministic path planner. Given a particular input, the randomized path planners return different outputs whereas the deterministic path planner will always produce the same output. Therefore, the behavior of search-based path planning is more predictable and easy to control. The search-based algorithm creates a motion-primitive subset that defines the movement of the mobile robot with several simple primitive motions and then repeatedly applies those trajectories during the path planning process to eventually create a feasible path.

2.2.1 Randomized Path Planner

The RRT and the PRM explore the space by randomly sampling the space. The RRT randomly samples the space to build a tree object that roots from the start and reaches the goal, whereas the PRM randomly samples the space to create a roadmap object that covers the start and the goal.

Rapidly Exploring Random Tree

The RRT [2, 64, 65] provides a new tool for mobile robot path planning. It is based on an open loop system model of a mobile robot. Since all the explored states of the RRT are generated based on the system model, it guarantees that any state in the exploring tree can be reached by following the nonholonomic constraints. This makes the RRT a useful tool for solving nonholonomic problems. As Fig. 2.7 shows, the blue and green objects represent the starting point and the goal respectively. The RRT can start from either one of the objects or both and uniformly explore the space until the goal has been reached.

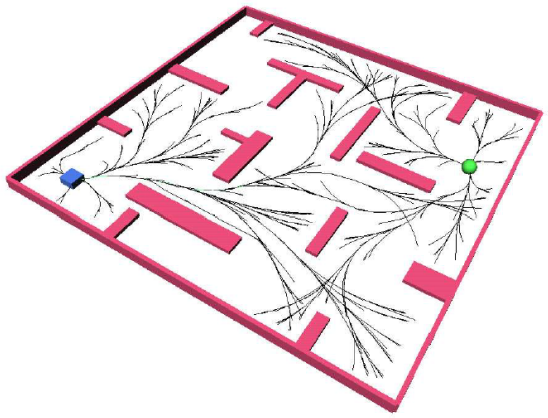


Figure 2.7: Rapidly-exploring random trees (image from [2])

The algorithm of the RRT is shown as Alg. 2.1, the first node of T is $\zeta_{init} \in S_{free}$ on the start state. In each iteration, a random state ζ_{rand} is selected from space S, and the closest node n_{near} to ζ_{rand} is found in line 5, and then an input u, which minimizes the distance from $n_{near}.\zeta$ to ζ_{rand}, is selected (line 6) and applied to the system to produce a new state ζ_{new} based on $n_{near}.\zeta$ (line 9). The new state ζ_{new} is added as a new node n_{new} to the exploring tree T (line 8). A traversable path is found if any new node lands on the goal (line 10). Otherwise the loop continues until $k = K$ (line 3), which means no path is found.

Algorithm 2.1: Generate RRT

ζ_{start}: Initial state of mobile robot
ζ_{goal} : Goal state
K :
T : Exploring tree object

```
1  Function ret Generate_RRT(ζ_start, ζ_goal, K, △t)
2      T.init(ζ_start);
3      for k=1 to K do
4          ζ_rand ← Random_State();
5          n_near ← Nearest_Neighbor(ζ_rand, τ);
6          u ← Select_Input(ζ_rand, n_near.ζ);
7          ζ_new ← New_State(n_near.ζ, u, △t);
8          n_new ← {ζ_new, n_near, u};
9          T.add_node(n_new);
10         if n_new.ζ = ζ_goal then          // when new node lands on ζ_goal
11             return n_new;                 // path is found, return last n_new
12     return null;                          // no path is found
```

[66] shows that the RRT will uniformly distribute over the space as the number of iterations approaching infinity. However, without any bias towards the goal, the convergence of the RRT is very slow. The RRT-GoalBias is an improved RRT-based planner deduced from redefining the function *Random_State*() (Alg. 2.1, line 4) with a function that tosses a biased coin to determine whether a random state or the goal state will be returned. With a small probability of the returned goal states, the RRT-GoalBias converges to the goal much faster but also brings the problem of the planner likely being trapped in the local minimum [66]. The RRT-GoalZoom made some improvement by choosing the random state in the whole space or the space around the goal. However, it still cannot avoid the planner being trapped due to the existence of the local minimum [66].

To avoid being trapped in the local minimum, the RRT-Connect [65] extends the step until the goal or an obstacle is reached. [67], [68] are such types of approaches; however, the extended steps seriously reduce the optimality of the produced path.

The bidirectional planner [66] and the local trees [69] planner maintain multiple trees simultaneously that can explore the free space very fast. It has proved effective where the planner has to repeatedly pass several different regions. However, as pointed out by [66], to grow multiple random trees is similar to the PRM [42]. It shows the same problem of the PRM which will be introduced in the following section.

On the one hand, the uniform distribution of the RRT can take too much time to reach the goal. On the other hand, greedy heuristic can make the distribution of the RRT more goal-oriented but can trap the planner in the local minimum. The abovementioned approaches mostly try to find a balance between them but both problems are not really solved. Another problem of the RRT is its randomness. Since the RRT randomly explores the space, the result is almost surely not optimal. This is also one of the problems of the PRM. Therefore, the produced path cannot be applied directly and extra optimization is always required, increasing the complexity and the computation cost.

Probabilistic Roadmap Method

The PRM [as shown in Fig. 2.4(a)] has been introduced by [42,70–74]. Basically the concept of PRM includes two phases—a learning phase, and a query phase.

During the learning phase (as in Alg. 2.2), the PRM randomly samples the search space and tests whether or not the samples are in free space (line 6). A roadmap is then created by repeatedly connecting each sample and its neighboring samples with a local planner (line 9). A sample is defined as a neighbor of the current sample if its distance from the current sample is less than a certain threshold. In the query phase (as in Alg. 2.3), the PRM tries to connect the start and the goal to the closest corresponding states ζ_s and ζ_g on the roadmap (line 3 and line 8) respectively, and then the path planning is done to search a path connecting ζ_s and ζ_g over the roadmap (line 12).

The PRM divides a global path planning problem into many local path planning problems since solving every single local path planning problem separately is much easier than directly solving the global path planning problem. This simplifies the process of global path planning.

Algorithm 2.2: Construct roadmap of PRM

V : Collection of samples
E : Collection of edges
$same\_connected\_component(\zeta', \zeta)$: Function test if ζ' and ζ in same component
$local\_planner(\zeta', \zeta)$: Function tries to find path between ζ' and ζ

```
1  V ← ∅;
2  E ← ∅;
3  loop
4      ζ ← random state in C_free;
5      V ← V ∪ {ζ};
6      N_c ← set of candidate neighbor configurations of ζ in V;
7      forall the ζ' ∈ N_c, in order of increasing distance from ζ do
8          if same_connected_component(ζ', ζ) = false then
9              if local_planner(ζ', ζ) ≠ null then
10                 E ← E ∪ {ζ'ζ}
```

Algorithm 2.3: Path query of PRM

ζ_{start}: Start state
ζ_{goal} : Goal state

```
1  Function ret Path_Query(ζ_start, ζ_goal)
2      V_s ← {ζ ∈ V | Distance(ζ_start, ζ) < M};
3      if planner can find a path between ζ_start and a state in V_s then
4          let ζ_s ∈ V_s be that state;
5      else
6          return failure;
7      V_g ← {ζ ∈ V | Distance(ζ_goal, ζ) < M};
8      if the planner can find a path between ζ_goal and a state in V_s then
9          let ζ_g ∈ V_g be that state;
10     else
11         return failure;
12     if a path P between ζ_s and ζ_g is found then
13         return overall solution path (ζ_start, ζ_s), p, (ζ_g, ζ_goal);
14     else
15         return failure;
```

The PRM has been used for many different motion planning problems [42, 75–81] because of its simplicity.

The PRM assumes that a roadmap can be constructed by connecting neighboring states. However, it is very difficult to connect two random states by following the nonholonomic constraints. This poses a challenge to constructing a roadmap for the PRM. In addition, the PRM also has problems when applied to narrow corridors connecting several subspaces. Since narrow corridors only take up very small spaces, there may not be enough samples to maintain the connectivity of the roadmap. This makes the PRM-based roadmap incomplete. The methods [82–87] proposed to solve this problem. However, for nonholonomic mobile robot path planning, any connection between sampled states must meet the nonholonomic constraints, otherwise the connection will no long be traversable for the robot. Finding a feasible connection between two neighboring states is non-trivial, which highly increases the computation cost of creating a PRM, making the PRM very sensitive to dynamic changes.

2.2.2 Search-based Path Planning Algorithms

Search-based algorithms use a set of motion primitives. The motion primitives (which will be covered in Section 2.4 later) define the most basic movement of the robot. By repeatedly making use of the motion primitives, the search-based algorithm grows a search tree that explores the space and eventually reaches the goal if a feasible path does exist. Unlike randomized algorithms that randomly explore the space, search-based algorithms incrementally explore the space based on the node with the minimum cost at every step, guaranteeing the optimality of the produced path.

Dijkstra's algorithm [88] is one of the first search-based path planning algorithms and uniformly explores every state in the space and returns the optimal path if it does exist. However, uniformly exploring the space can take too much computation time to reach the goal.

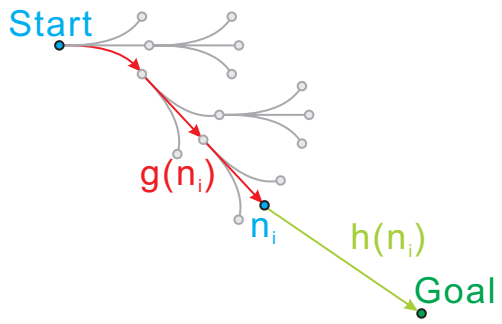


Figure 2.8: Exploring tree

The A* algorithm [89] introduces a heuristic value into the evaluation of each node in the search tree, which makes the search more goal-oriented, rather than uniformly exploring

the space. For the A* algorithm, each node in the exploring tree is evaluated based on the cost $f(n_i)$ calculated based on (2.1),

$$f(n_i) = g(n_i) + h(n_i) \tag{2.1}$$

where $f(n_i)$ is the summation of two parts—the path cost $g(n_i)$, and the heuristic cost $h(n_i)$. As shown in Fig. 2.8, the path cost $g(n_i)$ is the actual cost from the start to the current node (marked as red path), and the heuristic cost $h(n_i)$ is the estimating cost from the current node to the goal. Dijkstra's algorithm could be considered a special case of the A* algorithm, where $h(n_i)$ is constantly set as zero. The node with the smaller cost value $f(n_i)$ is regarded as more promising so it has more chances of being explored first. The A* algorithm iteratively explores the node with the lowest cost value $f(n_i)$. As in Fig. 2.8, the Euclidean-based heuristic cost helps the exploring tree always to grow towards the goal, instead of uniformly exploring the whole space, which saves both time and memory cost.

The A* algorithm is shown as Alg. 2.4, in which the search builds a search tree that is rooted to the start ζ_{start} (line 2). The search tree iteratively stretches out to reach the goal ζ_{goal} by exploring the space. The nodes of the search tree are saved in two sets—*openset*, and *closedset*. *closedset* keeps the nodes that have already been fully explored. *openset* keeps track of the nodes that have been reached, but have yet to be explored.

The complexity of the search in each step is reduced by narrowing down the candidate nodes in *openset* instead of the whole search tree. $g\_score[]$ saves minimum path costs where the search tree has currently achieved. In every step, the node n_c with the lowest cost in *openset* will be explored (line 10). Each possible neighbor n_i of n_c is validated, by comparing the path cost new_g_score of n_i with the value that saved at $g\_score[n_i.\zeta]$ (line 19), where $n_i.\zeta$ represents the state or location of n_i. If new_g_score is less than the value saved by $g\_score[n_i.\zeta]$ (line 19), it means n_i has made a better achievement, so $g\_score[n_i.\zeta]$ will be updated by new_g_score (line 21), and n_i will be added into *openset* (line 24) if it is still not in *openset*.

The search goes on until one of the nodes n_{goal} lands on the goal (Alg. 2.4, line 11), and then a feasible path can be found with Alg. 2.5 by traversing the search tree back to n_{start} from n_{goal}. If no node has ever landed on the goal until *openset* is empty (line 9), it means a feasible path does not exist.

Since each step is based on the node with the lowest cost calculated based on (2.1), the nodes with similar path costs $g(n_i)$ but lower heuristic costs $h(n_i)$ will always be explored first. Heuristic guides the search to explore the areas regarded more promising.

2.3 Heuristic

Based on the above analysis, the integration of the heuristic makes the search more goal-oriented, instead of uniformly exploring the whole space. This consequently reduces the computation cost. However, if the heuristic function $h(n_i)$ does not correctly reflect the true cost between n_i and the goal, the search can be guided to the wrong place and, sometimes,

Algorithm 2.4: A* algorithm

openset : Open Set
closedset : Closed Set
$neighbor\_nodes(n_c)$: Function returns the set of neighbor nodes around n_c

```
1  Function ret A_Star(ζ_start, ζ_goal)
2      n.ζ ← ζ_start;
3      n.parent_node ← null;
4      initiate all state in g_score[] Infinite;
5      g_score[n.ζ] ← 0;
6      n.f ← g_score[n.ζ] + heuristic(n.ζ, ζ_goal);
7      openset ← {n};
8      closedset ← ∅;
9      while openset ≠ ∅ do
10         n_c ← node in openset with lowest f cost;
11         if n_c.ζ = ζ_goal then
12             return reconstruct_path(n_c);
13         openset ← openset − {n_c};
14         closedset ← closedset ∪ {n_c};
15         forall the node n_i ∈ neighbor_nodes(n_c) do
16             if n_i ∈ closedset then
17                 continue;
18             new_g_score = g_score[n_c.ζ] + path_cost(n_c, n_i);
19             if n_i ∉ openset or new_g_score < g_score[n_i.ζ] then
20                 n_i.parent_node ← n_c;
21                 g_score[n_i.ζ] ← new_g_score;
22                 n_i.f ← g_score[n_i.ζ] + heuristic(n_i.ζ, ζ_goal);
23                 if n_i ∉ openset then
24                     openset ← openset ∪ {n_i};
25     return failure;
```

Algorithm 2.5: Path reconstruction

```
1 Function ret reconstruct_path(n)
2     path ← n;
3     n ← n.parent_node;
4     while n.ζ ≠ null do
5         path ← path + n;
6         n ← n.parent_node;
7     return path
```

even be trapped in the local minimum. This is the major problem of the heuristic-based search and is expected to be solved in this work. Defining an accurate heuristic function becomes the key to solve the problem of local minimum (as discussed in Section 1.2).

2.3.1 Euclidean-based Heuristic

Despite the intensive research on search-based path planning algorithms, the topic of heuristic has barely been touched. The most widely used heuristic is Euclidean-based heuristic h_{euc}. h_{euc} is calculated based on (2.2),

$$h_{euc} = \sqrt{(x_i - x_{goal})^2 + (y_i - y_{goal})^2} \tag{2.2}$$

which is simply the Euclidean distance from the current location (x_i, y_i) to goal (x_{goal}, y_{goal}). The Euclidean-based heuristic always guides the search directly towards the goal. This idea is very intuitive under an environment with no local minima, as incrementally moving towards the goal will make the search reach the goal eventually. However, as stated in Section 1.2, since the Euclidean-based heuristic ignores obstacle constraints, the search of the Euclidean-based heuristic will be trapped in the presence of the local minimum. This effect becomes more dominant under a clustered environment. The local minimum is the fundamental problem of path planning by applying Euclidean heuristic. Under high dimensional state-space, especially, the search could be almost indefinitely trapped.

For real-time applications, the robot is required to find a feasible path in a very short time span. D* algorithm [90] is proposed based on A* algorithm. It is able to efficiently update the path when only small parts of the environment change since only the affected fraction of the search space required recomputation. However, this is based on the fact that a global path has already been produced, which is still generated with the original A* algorithm. The D* algorithm can efficiently deal with the dynamics in a varied environment, but cannot solve the problem of the local minimum. There are approaches similar to D*—D* Lite [91], and Delayed D* [92].

A number of anytime algorithms, such as the Weighted A*, the Anytime A* and the Anytime Repairing A* [11–16], make use of the fact that inflating the heuristic term with

$\epsilon > 1$ as in (2.3) may substantially expedite the search.

$$f(n_i) = g(n_i) + \epsilon h(n_i) \tag{2.3}$$

However, this reduces the optimality of the produced path. In a sense, Dijkstra and A* algorithm are special cases of anytime algorithms, where $\epsilon = 0$ and $\epsilon = 1$ receptively. The concept of anytime algorithms is to initially generate a suboptimal solution by inflating the heuristic term with $\epsilon > 1$. If there is still time left, anytime algorithms keep improving the result until the optimal path is found or the calculating time has lapsed. Inflating the heuristic term only helps under an environment consisting of shallow local minima; for the environments that consist of deep local minima, the search can nevertheless be trapped.

The Manhattan distance (2.4) can also be used as a heuristic because of its simplicity. However, since it still does not include any information regarding obstacle constraints, Manhattan distance-based heuristic h_{manh} also attracts the search directly toward the goal, which can inevitably lead the search to be trapped by the local minimum in the same way as the Euclidean-based heuristic.

$$h_{manh} = (x_i - x_{goal}) + (y_i - y_{goal}) \tag{2.4}$$

2.3.2 Reference Path-based Heuristic

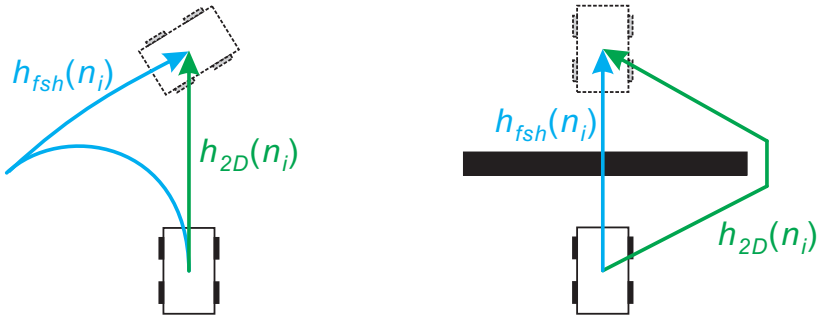
There are other approaches [18–21] that produce a reference path first with a simple planner without considering the nonholonomic constraints based on the GVD-based road map; then a second search tries to follow the reference path with a randomized path planner that follows the nonholonomic constraints. The reference path is used as guidance for the randomized path planner. As a result, the second search will be less likely to be trapped in the local minimum. Therefore, the reference path serves in the same way as heuristic. Sometimes, the reference path indeed helps speed up the search. However, since the reference path does not follow the nonholonomic constraints, it is not guaranteed optimal or not even traversable for the robot. For example, the reference path may pass through a narrow corridor with sharp turns that violate the nonholonomic constraints, which may still trap the search and take even longer than Euclidean-based heuristic.

2.3.3 Informative Heuristic

The informative heuristic was proposed by [93], which solves the problem by running a 2D $[(x, y)]$ version of Dijkstra's search starting at the cell of the goal. The 2D search computes the costs of shortest paths from the cell of the goal cell into all other cells in the environment. This search is therefore rerun every time the goal is changed. The costs are used later as part of the informative heuristic (2.5) to guide the motion primitive-based search in 3D $[(x, y, \theta)]$ or higher dimensional space.

$$h(n_i) = max(h_{fsh}(n_i), h_{2D}(n_i)) \tag{2.5}$$

As shown in Fig. 2.9, the green line represents obstacle-constrained cost $h_{2D}(n_i)$ of n_i that calculated by a 2D Dijkstra's search, and the blue line represents nonholonomic-constrained cost $h_{fsh}(n_i)$ of n_i regardless of obstacles. In Fig. 2.9(a), when n_i is close to the location of the robot without obstacles, there is $h_{fsh}(n_i) > h_{2D}(n_i)$, then $h(n_i) = h_{fsh}(n_i)$ [based on (2.5)]. However, when n_i and the robot are separated far apart by the obstacles in a



(a) (b)

Figure 2.9: a) Informative heuristic without obstacle, $h(n_i) = h_{fsh}(n_i)$; b) Informative heuristic with obstacle, $h_{fsh}(n_i) < h_{2D}(n_i)$

clustered environment as shown in Fig. 2.9(b), there is $h_{fsh}(n_i) < h_{2D}(n_i)$, then $h(n_i) = h_{2D}(n_i)$ [based on (2.5)]. This means that the informative heuristic only depends on obstacle-constrained cost $h_{2D}(n_i)$, regardless of nonholonomic constraints. $h_{2D}(n_i)$ helps to better lead and navigate the search through obstacles. However, as it will be shown in Section 7.2.2, the search can still be trapped, when it is guided to the narrow corridor that violates the nonholonomic constraints of the robot (this will be discussed more in detail later in Section 5.3). Additionally, obstacle-constrained cost $h_{2D}(n_i)$ of informative heuristic is required to recalculated every time the goal is changed, which becomes another overhead of the search.

2.3.4 GVD-based Solutions

The idea of applying the GVD to facilitate mobile robot path planning has been applied in the works of [18–21], which use a reference path (produced based on GVD) to guide the search. However, as discussed above in Section 2.3.2, they are unable to deal with the problem when the reference path passes through a narrow corridor that violate the nonholonomic constraints. [86, 87] are PRM-based path planning algorithm, which tries to make the samples in the neighborhood of the GVD and, therefore, covers the shortage of the PRM regarding narrow corridors. However, as mentioned in 2.2.1, connecting two neighboring states by following nonholonomic constraints is a non-trivial problem, this largely increases the computation cost of creating the PRM. Additionally, the randomized algorithm does not guarantee the optimality of the produced path, which becomes another issue of this approach.

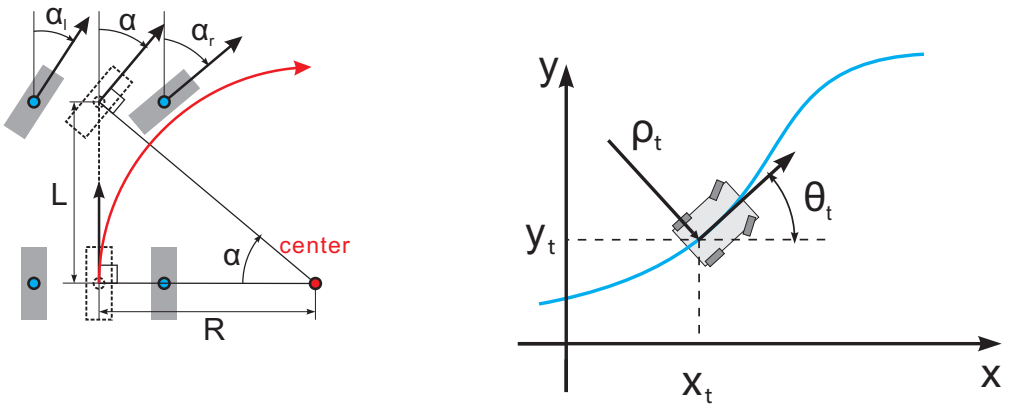
As it will be proposed in Chapter 5, the proposed Voronoi-based heuristic directly uses the entire GVD-based roadmap to guide the search, instead of a simple reference path. Additionally, the Voronoi-based heuristic adaptively varies during the process of path planning to avoid the search being trapped. This makes the proposed approach capable of dealing with both nonholonomic-constrained and obstacle-constrained local minimum with a small computation cost.

2.4 Motion Primitives

Motion primitives can be considered as a collection of outputs of the system by applying a set of sampled control input. More complex movements can be created with various combinations of motion primitives. Motion primitives simplify the problem of nonholonomic path planning since the nonholonomic constraints of the system are automatically integrated. The planner is allowed to ignore the nonholonomic constraints and focus on the obstacle constraints. In Chapter 3, motion primitives are used as a tool to solve the problem of nonholonomic constraints as discussed in Section 1.2.

2.4.1 Trajectory Requirements of Car-like Robots

Fig. 2.10(a) shows the Ackermann steering system [10] of the car-like robot. When the robot steers, its inside and outside front wheels are following the orbits of different radii. In order to perform a stable steering [as in Fig. 2.10(a)], different steering angles α_l and α_r need to be applied on the front wheels. The Ackermann steering system produces different steering angles for the front wheels to turn peacefully [10].



(a) (b)

Figure 2.10: a) Ackermann steering; b) Configuration space $(x_t, y_t, \theta_t, \rho_t) \in S^4$ of car-like robot

As shown in Fig. 2.10(a), the platform of a car-liker robot is simplified as a bicycle model [10] in order to simplify the geometry of the steering motion. This represents the front wheels and the rear wheels as the virtual wheels in the center, where L is the wheelbase of the car-liker robot, and α is the steering angle of the virtual front wheel. Based on the Ackermann steering geometrics in Fig. 2.10(a), given wheelbase L, turning radius R (or curvature ρ) depends directly on steering angle α calculated by (2.6).

$$R = L/tan(\alpha) \; or \; \rho = tan(\alpha)/L \tag{2.6}$$

As shown in Fig. 2.10(b), the search space of the car-like robot is a 4D configuration space $(x_t, y_t, \theta_t, \rho_t) \in S^4$, x_t and y_t represent the position of the robot, θ_t and ρ_t are the orientation and the turning curvature, respectively. Since the steering angle is limited by the maximum steering angle α_{max}, this results in a maximum turning curvature of the car-like robot $\rho_{max} = tan(a_{max})/L$. ρ_{max} is the major constraint for the car-like robot and is also the most distinct feature that differentiates the car-like robot from other types of mobile robots.

The steering angle α of the front wheels can only vary continuously. Based on (2.6), the trajectory curvature depends directly on steering angle α, so the curvature along the trajectory must also vary continuously. Therefore, from the above analysis, the curvature ρ along the trajectories of the car-like robot should meet the following conditions:

①: ρ must be continuous along the trajectory; and

②: $-\rho_{max} \leq \rho \leq \rho_{max}$

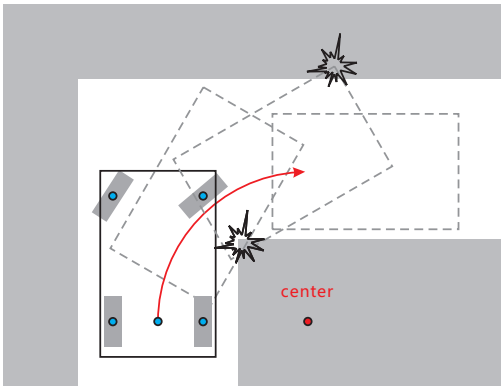


Figure 2.11: Nonholonomic constraints of car-like robot

As Fig. 2.11 shows, the robot is trying to go through a corridor and the red curve shows the trajectory of its maximum steering. Clearly, the corridor is no longer traversable for the car-like robot because of limited maximum steering curvature ρ_{max}. This special behavior makes car-like robot path planning much more complex since it has to take care of not only obstacle constraints but also nonholonomic constraints. The situation shown in Fig. 2.11 forms a very challenging task and appears very often in clustered environments. In order to avoid the robot being trapped in such situation, a new approach to detect and deal with this problem at a very small computation cost is proposed in Chapter 5.

2.4.2 Primitive Trajectory-based Path Planning

Fig. 2.12(a) is the result of the classic grid map-based path planning algorithm which defines the movement from the current cell to its eight neighbors as motion primitives [Fig. 2.12(b)]. However, this cannot be directly applied to nonholonomic mobile robot since the produced path based on the the grid map search consists of many sharp turns as shown in Fig. 2.12(a). Primitive trajectory-based path planning is applied [Fig. 2.12(d)] to solve this problem. The

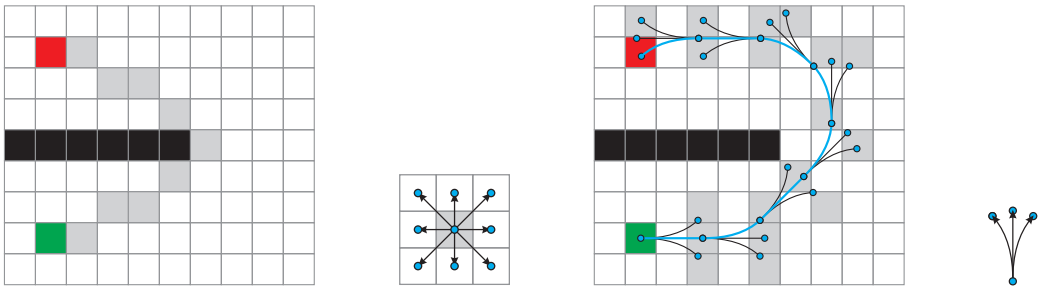


Figure 2.12: a) Grid map-based path planning; b) 8 neighbor cells; c) Primitive trajectory-based path planning; d) Primitive trajectory set

primitive trajectories are several simple possible trajectories that the robot can move in a very short step, which describes the basic movement of the nonholonomic mobile robot. As shown in Fig. 2.12(c), the primitive trajectory set has been repeatedly used to generate a continuous path. Details of the primitive trajectory set will be proposed in Chapter 3.

[9] applies Ariadne's Clew algorithm [94] as the planner which is a path planning algorithm similar to Dijkstra's algorithm [88]. [95] improved the above approach [9] by combining the A* [89] algorithm. As mentioned in last section, the car-like robot requires a continuous curvature along the path. Taking the curvature into account results in too many variations and makes it very difficult to find a trajectory set that is able to fully describe all movements of the car-like robot. In order to simplify the problem, [96] always assumes the curvature ρ at the start and the end point of any primitive trajectory is zero. With such a feature, different subtrajectories can be easily connected to each other without disrupting the continuity of the curvature. However, the flexibility of the produced path is largely limited. This approach becomes inadequate when more intricate maneuvers are required in a clustered environment. Although the state lattice [8] can produce more natural maneuvers, it requires a substantial computation in advance to produce a very complicated trajectory set. The computational expense of path planning over a long distance with the state lattice remains challenging making the state lattice planner more applicable to local path planning.

In Chapter 3, a new primitive trajectory set is proposed to make the produced path more flexible. Unlike the state lattice that tries to define a complete trajectories set with a large computation cost in advance, the proposed trajectory set is defined in a much simple

form. Additionally, the mentioned works have tried to produce a smoothed trajectory set, but not many of them have ever discussed how to minimize the variation of the curvature to avoid aggressive steering (as we stated in Section 1.2). This becomes another problem that Chapter 3 is supposed to solve.

2.5 Local Path Planning

Generally speaking, after the global path has been found, the mobile robot will try to follow the path and reach the goal. However, there are usually some dynamic objects moving around in the environment, and can block the pre-calculated path. The general solution would be combining global path planning with either local path planning or obstacle avoidance. Therefore, an effectively integration of the two processes (as mention in Section 1.2) becomes another problem that is expected to be solved in this dissertation.

2.5.1 Artificial Potential Field

An artificial potential field (APF) [97] treats the robot as a point in a potential field where the robot is assumed to be attracted to the goal and repulsed by the obstacles. This approach requires very little computation but it can easily be trapped in the local minima. Therefore, the APF is generally used as a local planner under the guidance of the global path [98–100]. The global path helps avoid the global static local minimum but it is still possible for the APF to be trapped by the local minimum formed by the dynamic obstacles.

2.5.2 Vector Field Histogram

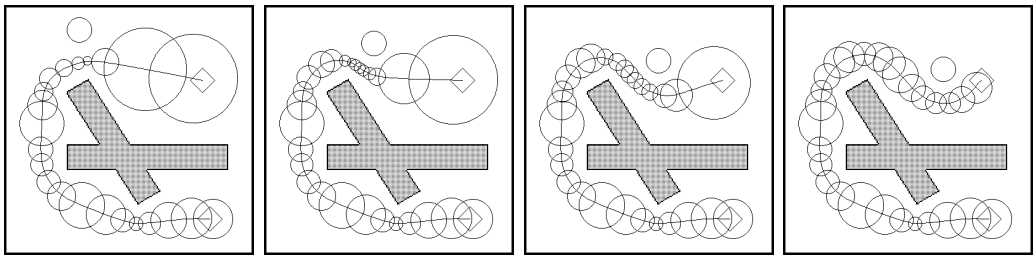
The vector field histogram (VFH) method builds a polar histogram based on the discetized directions around the robot's current position. The VFH assigns a cost value to each of these primary candidate directions, and selects the primary direction with the lowest cost as its new direction of the next movement [101–105]. However, the VFH only considers the immediate effects of its selections so, consequently, the VFH may also be trapped in the local minimum.

The problem of the APF and the VFH can be partly solved by setting intermediate goals along the global path. The robot can avoid the global local minima by sequentially reaching each intermediate goals. However, this still cannot avoid the local minimum formed by the dynamic obstacles. The VFH* [106] is proposed to solve these shortcomings of the VFH by constructing several histograms to form a search tree based on the A* algorithm. This is similar to locally applying primitive trajectories-based path planning (Section 2.4.2). However, since the search always tries to approach the intermediate goals and stay with the global path as much as possible, the global path becomes another type of constraint reducing the flexibility of local path planning as described in Section 1.2. This makes the produce path no longer optimal. Chapter 6 proposes a new heuristic type, rather than set a sequence

of intermediate goals which guides the local path planner to follow the global path without reducing its flexibility.

2.5.3 Elastic Bands

Elastic bands [3, 107–112] are also among the widely used local path planners. The global path is transformed into a sequence of bubbles that forms the elastic bands. The attraction between the bubbles pulls the bands together, whereas the obstacles apply virtual forces on them. The bubbles move incrementally to minimize the forces. The bubbles can also be inserted or removed to maintain the connectivity of the elastic bands. As shown in Fig. 2.13, the elastic bands are "pushed" away from the obstacles, making it very flexible to deal with the dynamic obstacles.

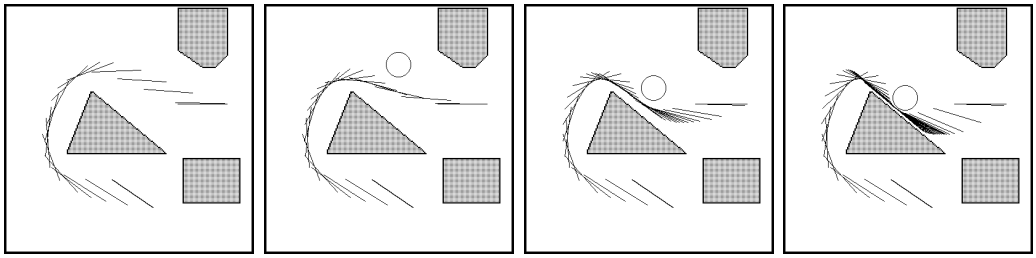


(a) (b) (c) (d)

Figure 2.13: "As the obstacle moves, the bubbles also move to minimize the force on the elastic band. If needed, bubbles are inserted and deleted to maintain a collision-free path." (image from [3])

One of the drawbacks of the elastic bands is that it can be "pushed" into a narrow space as the obstacles keep moving as in Fig. 2.14. In this scenario, the resulting path is no longer guaranteed safe since it is too close to the obstacles. The elastic bands can provide a very smooth path but the produced path is not guaranteed to follow the nonholonomic constraints. This becomes another problem of the elastic bands in terms of car-like robot path planning.

As mentioned, the above local path planners can be trapped in a certain the local minimum, especially under a clustered environment. The general idea is to set intermediate way points along the global path. The robot reaches the goal by approaching each intermediate way point sequentially with the local planner [22, 23, 113–116]. However, as introduced in Section 1.2, this can result in a snaky path when multiple obstacles are located in the original global path, since the robot has to simultaneously follow the global path and avoid obstacles in the way. As it will be shown in Section 7.4, this will dramatically reduce the optimality of the result. Therefore a new local planner will proposed to solve this problem in Chapter 6. The proposed local planner does not require the robot to follow any global path, which gives the robot enough flexibility to avoid obstacles while approaching the goal.



(a) (b) (c) (d)

Figure 2.14: The elastic bands can be "pushed"to a narrow area by the moving obstacles, even when there is more space on the other side of the obstacle (image from [3])

2.6 Summary

The current states of several aspects for car-like robot path planning concerning map representation, motion primitives, global and local path planning have been presented in this chapter. They play important roles in different sections of path planning. Any defect in them can reduce the optimality of the whole path planning result.

As mentioned in Section 2.1, there are two major types of map representations—metric maps, and topological maps. The metric map saves detailed information of the environment. It provides a straightforward connection to the real-world environment and is, therefore, very simple to create and modify. However, for large environments, the information contained by the metric maps is overwhelming. The planner directly applied to such maps can be easily trapped in the local area and take an almost indefinite amount of computation time to produce a feasible path. Topological maps contain highly compressed information of the space that connects the traversable subspaces with a very limited number of nodes. In order to maintain detailed information of the environment without losing the global perspective, the proposed path planning algorithm in this dissertation is based on a hybrid map structure of both metric and topological maps.

As discussed in Section 2.2, there are three major types of path planning algorithms for car-like robots. The RRT and the PRM are both randomized algorithms. Apart from their drawbacks mentioned before, the most severe shortcoming of randomized algorithms is the uncertainty of the produced path. In other words, with the same input, randomized algorithms can produce very different results due to the nature of randomized algorithms since both the RRT and the PRM explore the space by randomly sampling it. The uncertainty of the produced path makes the behavior of the RRT and the PRM very unpredictable. For real-time applications, this is a very dangerous feature. Search-based algorithms are deterministic algorithms that produce the same result when the inputs are the same. However the search-based algorithm also has its own drawbacks. As mentioned in Section 2.2.2, the computation cost directly depends on the accuracy of the heuristic function that estimates the distance from the current location to the goal. However, apart from the Euclidean

distance, the topic of heuristic for path planning has barely been touched. Since the Euclidean distance-based heuristic pulls the search directly towards the goal, the search could be trapped if there is any obstacle between the goal and the start. This is also the major problem that search-based algorithms might encounter under large, clustered environments. This problem is solved by a new type of Voronoi-based heuristic proposed later in this dissertation.

Motion primitives are another important part of search-based path planning. As discussed in Section 2.4, the existing approaches try to either develop a set of simple primitives that only defines very limited behavior of the car-like robot and, hence, reduces the diversity of the produced path, or define a complex primitive set that covers all possible movements although this requires a great amount of computation cost done in advance. A new primitive trajectory set that is simple to create but can still preserve the flexibility of the produced path is proposed in Chapter 3.

Given the presence of the dynamic obstacles, the produced path can become less optimal or even blocked in real time. In an environment that is full of such obstacles, the produced global path can almost never be applied directly. As introduced in Section 2.5, the existing approaches try to deal with this problem by partly modifying the global path in real time with the local path planner, i.e. the robot leaves the global path shortly and moves back later. However, as discussed in Section 1.2, apart from the obstacle and noholonomic constraints, the global path imposes extra constraints that the robot must follow. This reduces the flexibility of local path planning and can make the search trapped in the local minimum, which increases the computation cost and reduces the real time performance. In Chapter 6, the proposed local path planner avoids such constraints by converting the global path into a path corridor. The robot is free to move inside the path corridor and no longer required to strictly follow the global path.

To summarize, due to the inaccuracy of the heuristic, the current solutions for nonholonomic mobile robot path planning can easily drop into local minimum and require a high computation cost. This makes them either only applicable to small and simple environments, or not flexible enough to cope with the dynamics. A new type of heuristic will be proposed in the following parts of this dissertation, which enables the nonholonomic path planning process to be applied to large-scale, clustered dynamic environments with a very low computation cost.

Part II

Nonholonomic Constraints

Chapter 3

Primitive Trajectory Set for Car-like Robots

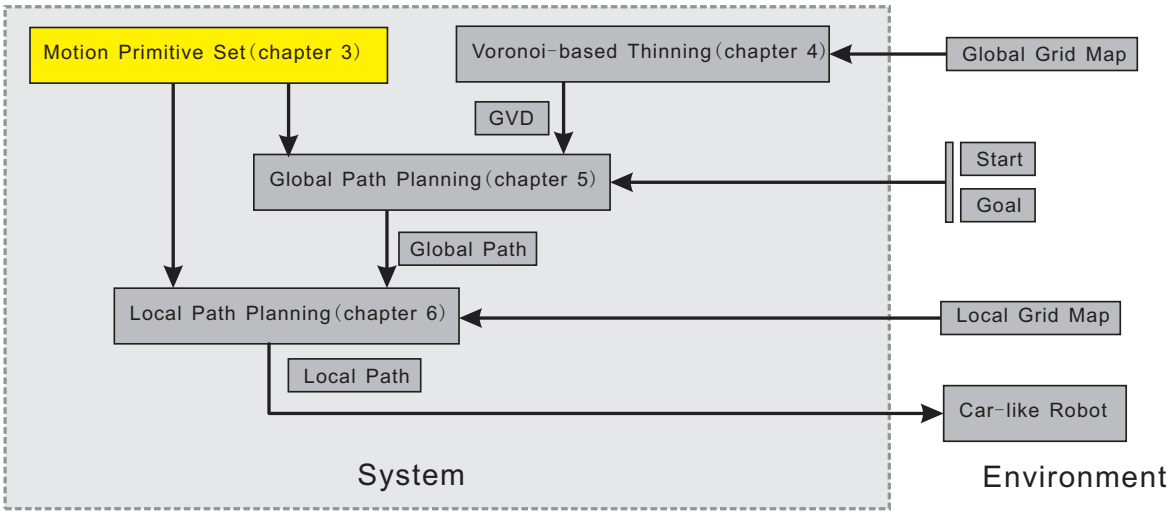
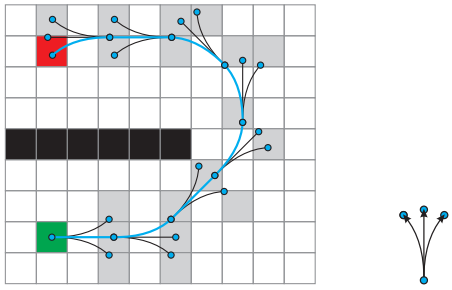


Figure 3.1: New Voronoi-based path planning process: primitive trajectory set

In this chapter, a very simple primitive trajectory set for car-like robots is proposed. As introduced in Section 2.4, primitive trajectories are a set of basic movements that the robot can perform in short steps [as shown in Fig. 3.2(b)], and can be combined to construct more complex movements to achieve the goal [Fig. 3.2(a)].

The primitive trajectory set (highlighted in Fig. 3.1) serves as the motion primitives of the car-like robot for both global and local path planning. The state lattice [8] defines the complex primitive trajectories that connect states $(x_i, y_i, \theta_i, \rho_i)$ in a 4D space, where θ_i and ρ_i represent the orientation and the curvature along the path respectively. The state lattice guarantees the continuity of curvatures along the path. However, as the search has to be



(a) (b)

Figure 3.2: a) Primitive trajectory-based path planning; b) Primitive trajectory set

done in a 4D space, the computation cost of the search is very high. [96] uses simplified trajectories that assume zero curvatures on both ends of the trajectories. Therefore, the trajectories can be easily connected without breaking the continuity of the curvatures and allows the search to be done in a 3D space. This largely reduces the computation cost. However, the assumption of zero curvature on both sides of the trajectory largely reduces the flexibility of the produced path, thus disturbing the optimality of the result.

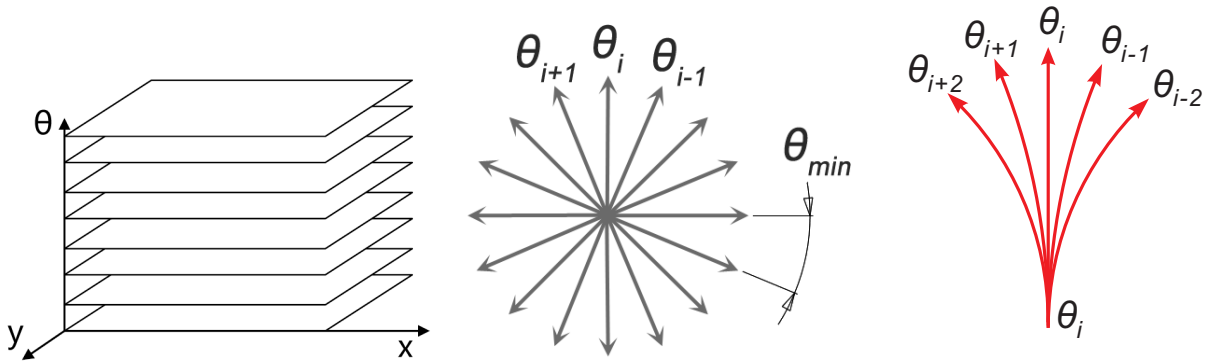
A different solution is proposed in this chapter. Unlike [8] that defines a complex primitive trajectory set, the proposed primitive trajectory set is comprised of only a few short primitive trajectories of constant curvatures (Section 3.1). However, they have unlimited combinations. This maintains the diversity of the movement that the robot can perform. The produced path is then smoothed using a Bézier spline [24] (Section 3.2) to meet the continuity of the curvatures. This allows the search to be done in a 3D space without considering the continuity of the curvatures, which, therefore, renders the search more flexible. As the trajectories are created based on the kinematic model of the car-like robot, the produced path naturally inherits all the nonholonomic requirements. The proposed primitive trajectory set is easy to generate and reduces the offline computation cost.

Additionally, the steering rate cost is proposed and integrated with the evaluation of the trajectories, which effectively reduces the variation of the curvature along the produced path. This allows the robot to steer peacefully while it moves along the produced path and largely avoids aggressive steering (as mentioned in Section 1.2).

3.1 Primitive Trajectory Set

As mentioned, the proposed primitive trajectories are defined in a discrete 3D space $(x_i, y_i, \theta_i) \in S_d^3$ as in Fig. 3.3(a) and 3.3(b). The directional dimension θ is divided in K directions, so $\theta_{min} = \dfrac{2\pi}{K}$ is the minimum step in directional dimension. As in Fig. 3.3(b), there is $\theta_i \in \Theta$, where Θ is the discrete directional space $\{\theta | \theta = k\theta_{min}, k \in \mathbb{Z}\}$.

As in Fig. 3.3(c), every primitive trajectory starts and ends in the discrete directional



(a) (b) (c)

Figure 3.3: a) 3D dimensional space; b) Directional dimension with θ_{min} as minimum step; c) Primitive trajectories always start and end inside the space S_d^3

space Θ. The proposed primitive trajectory set consists of two subsets, the basic and extended set. The basic set defines the primary movement of the car-like robot, and then it is extended to a more general form (extended set) to produce more natural and further smoothed trajectories.

3.1.1 Basic Primitive Trajectory Set

Fig. 3.4 shows a simplified bicycle model for the car-like robot [10] which is mentioned in Section 2.4.1 [Fig. 2.10(a)]. The front and rear wheels are represented as the virtual wheels in the center (shown as the gray boxes in Fig. 3.4). As shown in Fig. 3.4(a), the wheelbase is set as L, and the maximum steering angle α_{max} represents the maximum steering limit of the front wheel. The steering space is divided in M sections, so the minimum steering step is $\alpha_{min} = \dfrac{2\alpha_{max}}{M}$.

The primitive trajectory T_α is shown as the red curve in Fig. 3.4(b). T_α is the path on which the robot moves with the steering angle $\alpha = h\alpha_{min}(h \in \mathbb{Z})$. In order to make T_α always end in the discrete directional space Θ, T_α terminates when the orientation of the robot turns $\Delta\theta = h\theta_{min}$ as shown in Fig. 3.4(b), where θ_{min} is the minimum step in the discrete directional dimension [as in Fig. 3.3(b)].

Based on the geometry shown in Fig. 3.4(b), the turning radius of T_α is $R_\alpha = L/tan(\alpha)$, so the length Δs_α of $T_\alpha(\alpha = h\alpha_{min}, h \neq 0)$ could be calculated as (3.1).

$$\Delta s_\alpha = R_\alpha \Delta\theta_\alpha = \frac{L}{tan(\alpha)} h\theta_{min} = \frac{L}{tan(h\alpha_{min})} h\theta_{min} \qquad (\alpha = h\alpha_{min}, h \neq 0) \qquad (3.1)$$

However, when the steering angle $\alpha = 0$ $(h = 0)$, there are $R_\alpha = \infty$ and $\Delta\theta_\alpha = 0$. Therefore the length Δs_0 [Fig. 3.4(c)] of the trajectory $T_\alpha(\alpha = 0)$ cannot directly be

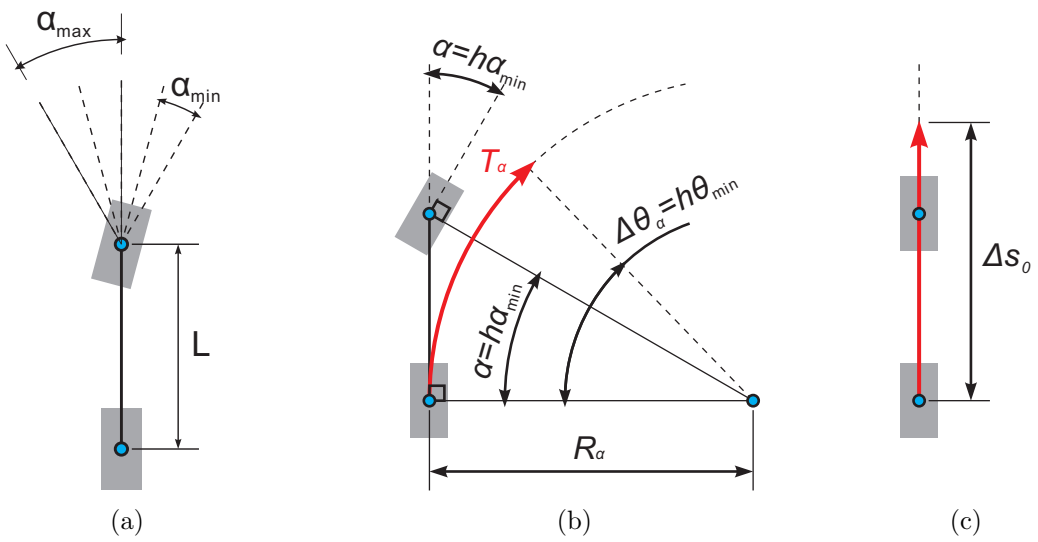


Figure 3.4: a) Steering space division; b) Primitive trajectory T_α with steering angle $\alpha = h\alpha_{min}$; c) Primitive trajectory T_α with steering angle $\alpha = 0$

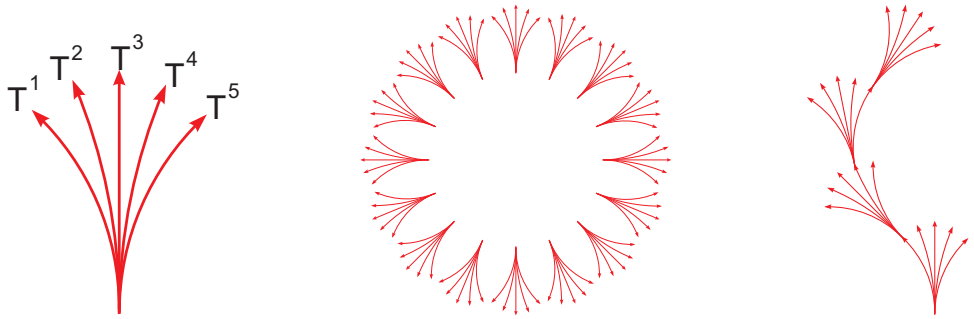
calculated with (3.1). Since h cannot be directly set as 0, we substitute h in (3.1) with an indefinitely small value $\epsilon \in \mathbb{R}$, then Δs_0 can be calculated by taking the limit of Δs_α with $\epsilon \to 0$ as (3.2).

$$\Delta s_0 = \lim_{\alpha \to 0} \Delta s_\alpha = \lim_{\epsilon \to 0} \frac{L}{tan(\epsilon \alpha_{min})} \epsilon \theta_{min} \tag{3.2}$$

When $\epsilon \to 0$, then $tan(\epsilon \alpha_{min}) \to \epsilon \alpha_{min}$, so $tan(\epsilon \alpha_{min})$ in (3.2) can be substituted with $\epsilon \alpha_{min}$, which yields (3.3).

$$\Delta s_0 = \lim_{\epsilon \to 0} \frac{L}{\epsilon \alpha_{min}} \epsilon \theta_{min} = \frac{L \theta_{min}}{\alpha_{min}} \tag{3.3}$$

As mentioned at the beginning of this chapter, unlike [8], a very simple primitive trajectory set will be proposed here. The total steering space is divided into only four basic steering sections, there is $M = 4$. The minimum steering step angle is $\alpha_{min} = 2\alpha_{max}/4$, where α_{max} is the maximum steering angle of the car-like robot. The trajectories in the basic primitive trajectory set are created by setting the steering angle $\alpha = \pm 2\alpha_{min}, \pm \alpha_{min}$ and 0, respectively, as shown in Fig. 3.5(a). Trajectory T^3 is generated by applying no steering. Trajectories T^1 and T^5 are generated by applying steering $-2\alpha_{min}$ and $2\alpha_{min}$, which terminate when the orientation turns $-2\theta_{min}$ and $2\theta_{min}$, respectively. Trajectories T^2 and T^4 are generated by applying steering $-\alpha_{min}$ and α_{min}, which terminate when the orientation turns $-\theta_{min}$ and θ_{min}, respectively..



(a) (b) (c)

Figure 3.5: a) Basic primitive trajectory set; b) Full directional basic primitive trajectory set; c) Simple combination of trajectories

In order to reduce calculations during the search, the trajectory set is rotated into all K directions in advance [Fig. 3.5(b)]. Fig. 3.5(c) shows a resulting path constructed by duplicating the basic primitive trajectories [Fig. 3.5(c)].

3.1.2 Extended Primitive Trajectory Set

In the basic primitive trajectory set, $\alpha_{max}/2$ is the minimum steering angle for the robot to steer. However, when the robot is moving at a higher speed, the steering angle is required to be small, or else the robot can lose its stability or even roll over. To solve this issue, the steering angle is further subdivided into smaller angles by taking the half of the original amount $\alpha_{max}/2$ [as in Fig. 3.6(a)]. Fig. 3.6(b) and 3.6(c) are the extended primitive trajectories by applying steering $\alpha_{max}/4$ and $\alpha_{max}/8$ respectively. The extended trajectory terminates when the body of the robot turns θ_{min}. For even finer steering angles, more divisions with a smaller steering angle $\alpha_{max}/2^m$ can be created accordingly. By combining with the basic primitive trajectory set, the full primitive trajectory set can be obtained [Fig. 3.6(d)]. Various types of maneuvers can be produced by applying different combinations of the proposed primitive trajectories.

3.2 Smoothed by Means of Bézier Spline Fitting

As mentioned in Section 2.4.1, the curvature ρ along the trajectories of the car-like robot should meet the following conditions:

①: ρ must be continuous along the trajectory; and
②: $-\rho_{max} \leq \rho \leq \rho_{max}$

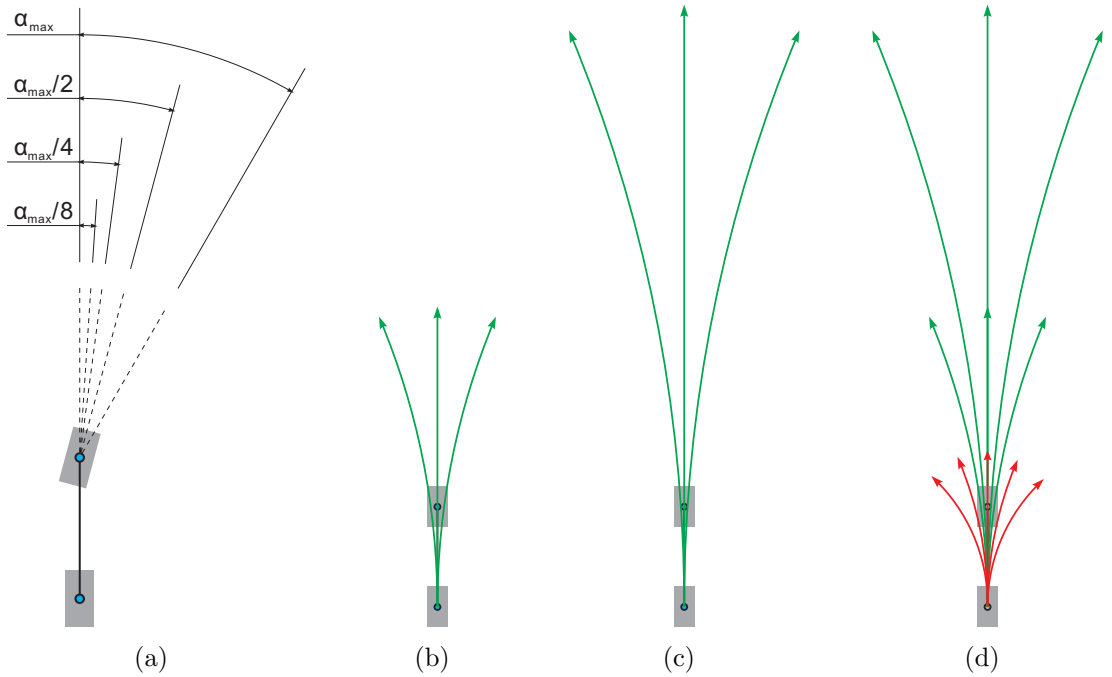


Figure 3.6: a)Extended division of steering angle; b) Extended primitive trajectories of steering angle $\alpha = \alpha_{max}/4$; c) Extended primitive trajectories of steering angle $\alpha = \alpha_{max}/8$; d) Full trajectory set

The path generated with this primitive trajectory set guarantees that the curvature ρ along the path never goes beyond the limit $-\rho_{max} \leq \rho \leq \rho_{max}$, i.e. the condition ②. However, ρ is not continuous at the joints of different primitive trajectories, and this still violates condition ①. To solve this problem, the proposed approach is to smooth the produced path with the Bézier spline [24]. Bézier spline is a parametric curve widely used in computer graphics to model smooth curves [117].

In Fig. 3.7, the black curves represent the primitive trajectories. The red curve is the Bézier spline which passes through each end point of the primitive trajectories n_0,n_1 and n_2. The curvatures of the primitive trajectories connecting n_0, n_1 and n_2 are ρ and $-\rho$ respectively. The initial directions and curvatures of both ends of the Bézier spline at n_0 and n_2 are defined the same way as that of the primitive trajectories. The purple box stands for the car-like robot, whereas the dashed lines outside the box represent the safety margin of the robot.

As shown in Fig. 3.7, the blue lines derived from the Bézier spline help visualize the curvature variation along the Bézier spline, which shows that the resulting Bézier spline has a continuous curvature. The Bézier spline is only slightly different from the original primitive trajectories. Fig. 3.7 shows the Bézier spline gives nice continuous curvatures without destroying the result of the original path.

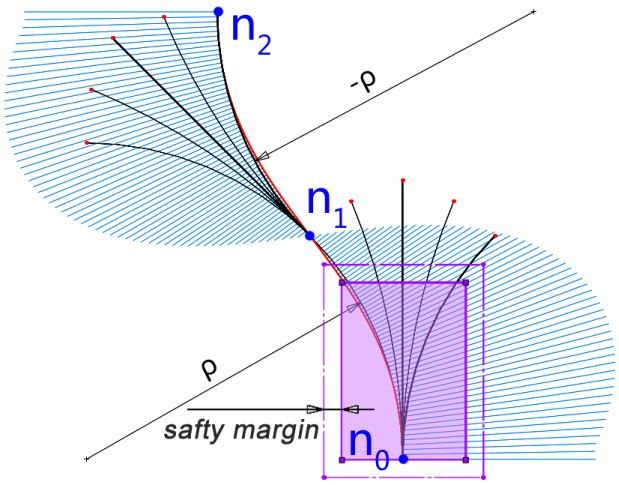


Figure 3.7: Smoothed with Bézier spline. (the blue lines derived from the Bézier spline help visualize the curvature variation along the Bézier spline.)

3.3 Path Cost Function

To evaluate the costs of primitive trajectories, a cost function $path\_cost(n_{i-1}, n_i)$ is proposed in this section. $path\_cost(n_{i-1}, n_i)$ is defined as the estimated cost for the robot to move from n_i to n_{i-1}, where n_i is one of the sub nodes of n_{i-1} connected by one of the primitive trajectories. Since an optimized trajectory should let the robot achieve the goal sooner, $path\_cost(n_{i-1}, n_i)$ is defined as the estimated minimum time cost while the robot moves from n_i to n_{i-1}. $path\_cost(n_{i-1}, n_i)$ is used later in the A* algorithm to find the path with the minimum time cost from the start to the goal.

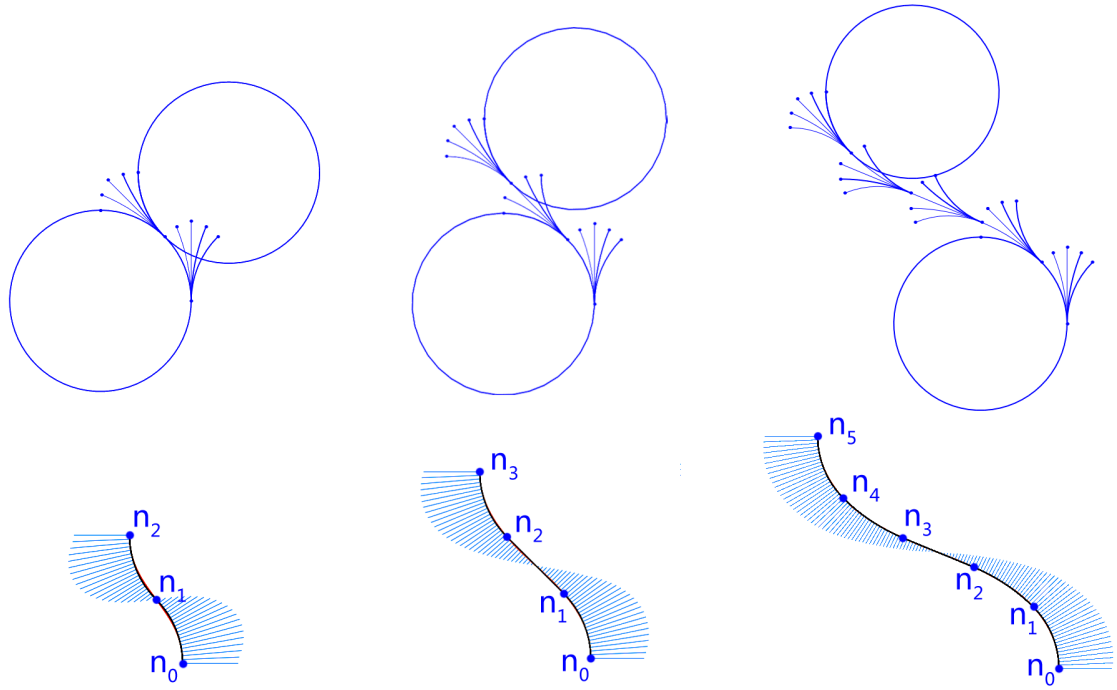
The proposed cost function $path\_cost(n_{i-1}, n_i)$ consists of two factors—steering rate cost, and the clearance cost. They evaluate the produced path from different perspectives. The steering rate cost attaches less cost to the path of less steering. The steering rate cost makes the produced path to be as smooth as possible which can effectively avoid the aggressive steering mentioned in Section 1.2. The clearance cost attaches less cost to the path that is relatively far away from the obstacles. The clearance cost avoids the produced path to be too close to the obstacles.

3.3.1 Steering Rate Cost

The maximum steering rate ω_{max} is the maximum steering velocity, which defines how fast the front wheels can steer. ω_{max} can affect the linear velocity of the robot. This can be explained with Fig. 3.8, which shows different steering strategies. Fig. 3.8(a) shows the combination of the trajectory set that is required to steer from the extreme left position $\alpha = -\alpha_{max}$ to the extreme right position $\alpha = \alpha_{max}$. In such case, the car-like robot has to

either apply large-scale steering in a very short time (aggressive steering) or reduce its linear velocity to slow down the steering. Frequently applying aggressive steering can damage the steering system, whereas reducing the linear velocity increases the time cost of reaching the goal.

In Fig. 3.8(b), rather than steering directly from the extreme left to the extreme right, the robot steers from the extreme left to the center and then to the extreme right. The total steering angle is still the same, but the traveled distance is longer. With the same maximum steering rate ω_{max}, the robot is allowed to move faster on the path [as shown in Fig. 3.8(b)]. Fig. 3.8(c) is the third example in which the robot steers even slower. This results in a further smoothed trajectory and allows the robot to move faster.



(a) (b) (c)

Figure 3.8: Different steering strategies

Steering rate cost Δt_{steer} in (3.4)is the time cost to steer the front wheels from α_{i-1} to α_i. α_{i-1} and α_i are the expected steering position on n_{i-1} and n_i respectively. The total time cost of the robot to move from n_{i-1} to n_i should no less than Δt_{steer}, otherwise the robot could not steer to α_i on time, as the robot approaches n_i from n_{i-1}. To simplify (3.4), ω_{max} is transformed into steering rate coefficient $\kappa_{steer} = 1/\omega_{max}$, which is used to transfer the steering angle into the steering rate cost.

$$\Delta t_{steer} = \frac{\mid \alpha_i - \alpha_{i-1} \mid}{\omega_{max}} = \kappa_{steer} \mid \alpha_i - \alpha_{i-1} \mid \tag{3.4}$$

3.3.2 Clearance Cost

Out of the safety issues, the mobile robot is always expected to stay away from the obstacles or slow down as it approaches them. Based on this assumption, the trajectories too close to the obstacles can take longer to execute.

To evaluate the trajectories based on their distances from the obstacles, each primitive trajectory between n_i and n_{i-1} is further sampled with a sequence of trajectory points Pt_j, that are equally spaced at very small distances from each other (shown as the blue dots on the trajectory in Fig. 3.9). It is defined that $dist(Pt_j, Pt_{j-1})$ is the distance between Pt_j and Pt_{j-1}, k is the number of sampled points on the trajectory between n_{i-1} and n_i, v_j is the safety velocity that the robot travels from Pt_{j-1} to Pt_j. So, the total time cost that the robot travels from n_{i-1} to n_i can be calculated as (3.5).

$$\sum_{j=2}^{k} \frac{dist(Pt_j, Pt_{j-1})}{v_j} \tag{3.5}$$

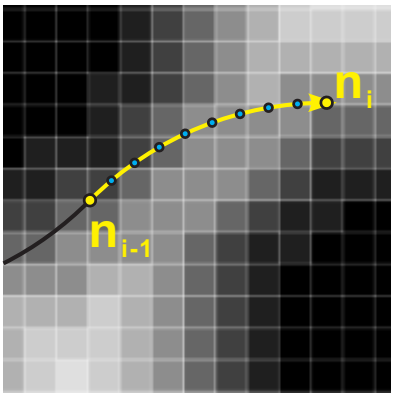


Figure 3.9: Evaluation of trajectory with sampled trajectory points on clearance map

As mentioned above, the robot is expected to slow down as it approaches the obstacles. Therefore, the safety velocity v_j on Pt_j is supposed to be very low when Pt_j on the primitive trajectory is close to the obstacles. Therefore, v_j is set proportional to the distance between Pt_j and its nearest obstacle, and such distance information is available from the clearance map. As mentioned in Section 2.1.3, each cell on a clearance map is assigned with a value representing the distance to its nearest obstacles. As shown in Fig. 3.9, the gray cells represent the distance information from the current cell to the obstacles. The darker it is, the closer it is to the obstacles.

Therefore, v_j is set proportional to the clearance value of Pt_j. There is $v_j = \kappa' \, clear[Pt_j]$. $clear[Pt_j]$ represents the clearance value of Pt_j on the clearance map, and κ' is the coefficient that transfers the clearance into a velocity. Since the sampled points Pt_j are equally spaced along the trajectory, the distance $dist(Pt_j, Pt_{j-1})$ between Pt_j and Pt_{j-1} can be defined as

a constant Δd. By substituting both v_j and $dist(Pt_j, Pt_{j-1})$ in (3.5) with $\kappa'\ clear[Pt_j]$ and Δd, respectively, the time cost affected by the clearance can be calculated with (3.6).

$$\Delta t_{clear} = \sum_{j=2}^{k} \frac{\Delta d}{\kappa'\ clear[Pt_j]} = \frac{\Delta d}{\kappa'} \sum_{j=2}^{k} \frac{1}{clear[Pt_j]} \tag{3.6}$$

Since both Δd and κ' are constant, Δd and κ' can be further combined together into a single clearance coefficient $\kappa_{clear} = \dfrac{\Delta d}{\kappa'}$; so, (3.6) becomes (3.7).

$$\Delta t_{clear} = \kappa_{clear} \sum_{j=2}^{k} \frac{1}{clear[Pt_j]} \tag{3.7}$$

3.3.3 Path Cost Function

As mentioned above, Δt is the minimum time cost for the robot to move from n_{i-1} to n_i. If Δt is less than Δt_{steer}, the robot fails to steer fast enough and can no longer stay on the path. If Δt is less than Δt_{clear}, it means the velocity of the robot goes beyond the safety limit regarding the obstacles. Therefore, the minimum time cost Δt must be no less than both Δt_{steer} and Δt_{clear}.

Apart from Δt_{steer} and Δt_{clear}, Δt also must be no less than Δt_{min}. Δt_{min} is the time cost while the robot moves on the trajectory with its maximum velocity. Therefore, Δt should be no less than any Δt_{steer}, Δt_{clear} and Δt_{min}. The path cost $path\_cost(n_{i-1}, n_i)$ between n_{i-1} and n_i is then defined as (3.8):

$$path\_cost(n_{i-1}, n_i) = \Delta t = maximum(\Delta t_{steer}, \Delta t_{clear}, \Delta t_{min}) \tag{3.8}$$

3.4 Footprint

The footprint is the trace that the mobile robot leaves as it moves along a trajectory. Each trajectory must be checked to see whether it is collision-free before it is applied. This means that the footprint of the trajectory cannot overlap any area labeled as an obstacle. For a circular mobile robot, this can be done by simply checking whether any of the points on the trajectory is too close to the obstacles. That is one of the reasons why the circular mobile robot is the most popular type. In the case of the car-like robot, the process is not as simple. This increases the complexity of the collision checking for the car-like robot. Some approaches [19] simplify the footprint to polygons. If the footprint polygon has overlapped any obstacle, then the trajectory is declared as not collision-free. However, this is only applicable to the polygon-based map rather than the gridmap. Here, a simplified gridmap-based collision checking of [118] will be proposed.

3.4.1 Footprint Trajectory T_α^{foot} and Footprint Width W_α^{foot}

The footprint is defined based on the margin box of the car-like robot as in Fig. 3.10. Wheel base L and wheel gauge W define the distances between the wheels. Front margin M_{front}, rear margin M_{rear} and side margin M_{side} define the margin box that is occupied by the car-like robot. Out of the safety issues, the margin box could be set slightly larger than the actual size of the robot.

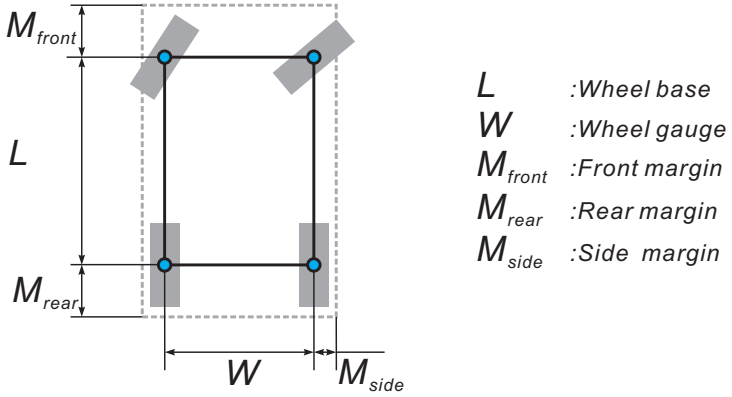
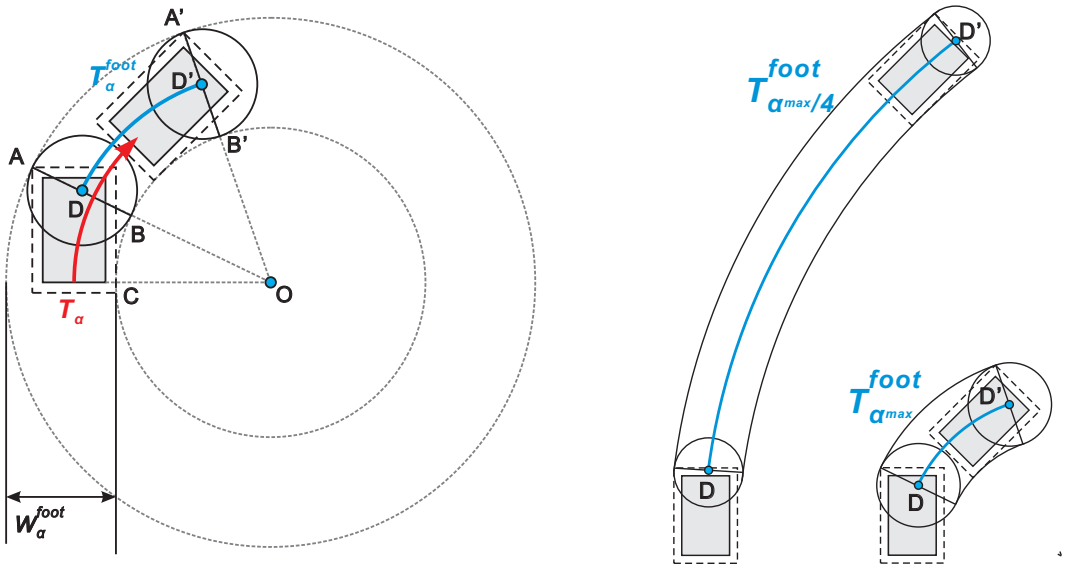


Figure 3.10: Margin box of the car-like robot

Fig. 3.11(a) shows a robot moving along the primitive trajectory T_α (the arrowed red line) around the center O. The gray box is defined by wheel base L and wheel gauge W, whereas the dashed box is the margin box of the robot. The circular track shows the footprint that a car-like robot will leave, according to the Ackermann steering mechanism [10] [Fig. 2.10(a)]. Points A and C are fixed to the robot, where point A is the top-left point of the margin box and point C is the bottom-right point of the margin box as shown in Fig. 3.10. The outside ring of the circular track is the trace of point A and the inside ring is the trace of point C. Both the outside ring and the inside ring form the footprint of the car-like robot. It is obvious that the width of the footprint W_α^{foot} is always wider than that of the robot. Point D is the intersection of line AO and the medial line of the footprint, D is also fixed with the robot. As the robot moves along the primitive trajectory, point D also moves along the medial line of the track to D', so the curve DD' forms the footprint trajectory T_α^{foot} of T_α. It is observable that primitive trajectory T_α can be considered as collision-free if there are no obstacles inside its footprint trajectory T_α^{foot} (DD').

Footprint trajectory T_α^{foot} is saved as a sequence of footprint trajectory points, which are spaced at very small distances from each other along T_α^{foot}. This greatly simplifies the collision checking of the trajectories. Clearly if any of the footprint trajectory points is too close to any of the obstacles, then trajectory T_α is considered as not collision-free.

Fig. 3.11(b) shows the footprint trajectories of different steering angles $\alpha = \alpha_{max}$ and $\alpha = \alpha_{max}/4$. It can be seen that smaller steering angles would also generate smaller W_α^{foot}.



(a) (b)

Figure 3.11: a) Footprint trajectory geometry; b) Difference of footprint trajectories with different steering angles

A little accuracy may be lost since there is a small overestimation at the end of the track section, but such a tiny loss is negligible in real applications.

3.5 Summary

A new primitive trajectory set for the car-like robot is proposed in this chapter. By combining and duplicating the primitive trajectories, various maneuvers can be created. The produced path is further smoothed with the Bézier spline which, in turn, results in a continuous maneuver. Instead of the distance cost, the time cost is used to evaluate the path. The time cost is affected by the steering rate, and the clearance cost. Large-scale steering in a short time range, especially with the steering rate cost, can be mostly avoided which produces more maneuvers that allow the car-like robot to move in a stable state.

Part III

Voronoi-based Path Planning

Chapter 4

Generalized Voronoi Diagram Extraction

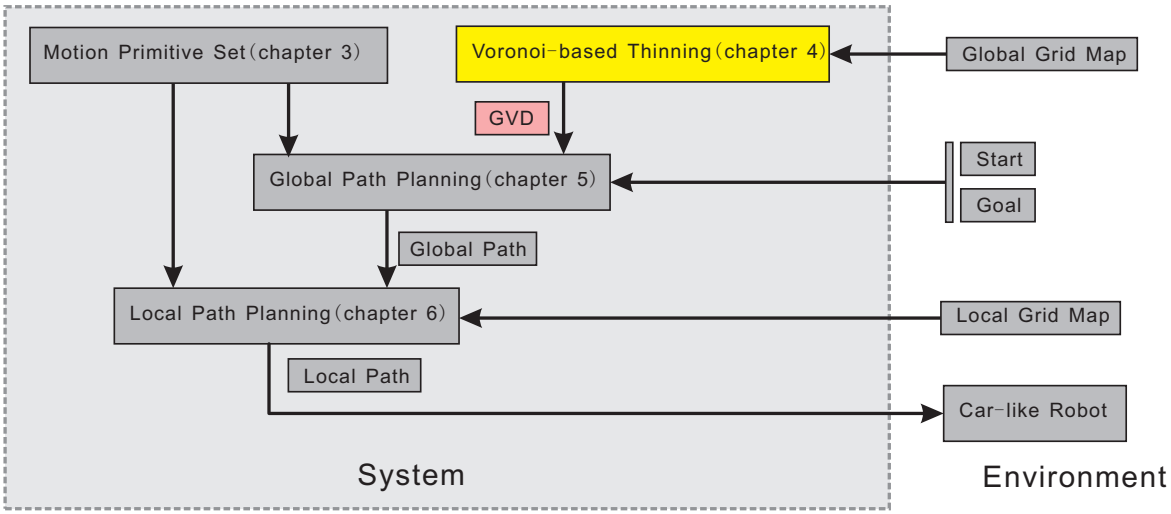


Figure 4.1: New Voronoi-based path planning process: Voronoi-based thinning

A topological map (introduced in Section 1.3) will be used to guide the search so that the search does not get trapped in the local minimum. The GVD is the ideal type of the topological map which can accurately reflect the geometry of the space (As discussed in section 2.1.2). However, an accurate GVD is always difficult to get. The method proposed in this chapter is to extract an accurate GVD from a given gridmap.

As discussed in Section 2.1.2, the thinning algorithm is easier to implement and requires less computation and memory cost compared to polygon map-based methods [49–52]. Since the thinning algorithm can be directly applied to the gridmap, it requires no conversion to the polygonal map. However, the thinning algorithm also has its own problems that may

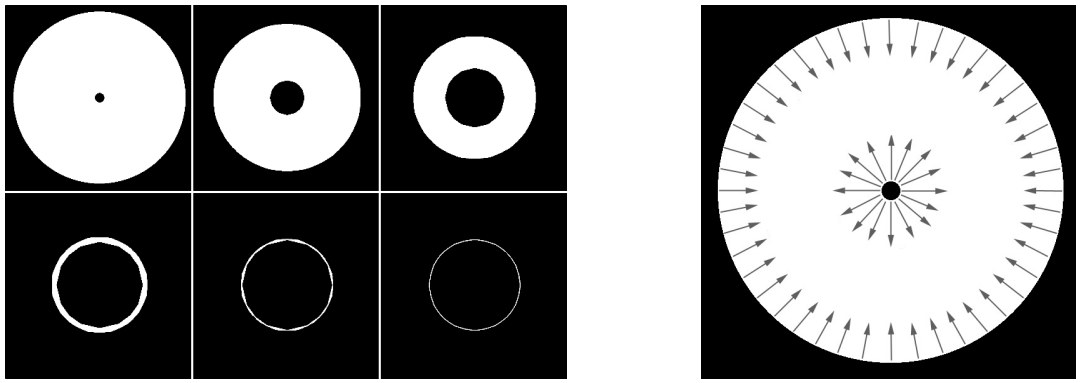
affect the accuracy of the produced GVD. Therefore, a new improved thinning algorithm will be proposed later in this chapter.

As the highlighted yellow box shown in Fig. 4.1, the proposed thinning process takes a gridmap as an input and produces the extracted GVD (shown as the pink box in Fig. 4.1) to serve the global path planning that will be covered in Chapter 5.

4.1 Ideal Thinning Algorithm

Fig. 4.2(a) shows the process of the thinning algorithm, in the top-left image, there is a white disk object with a hole in its center. As the following images show, the disk gets thinner and thinner until only a white circle is left.

In theory, an ideal thinning process spreads in all directions with the same speed and results in the medial line of the object [Fig. 4.2(b)]. However (as Section 4.4 will discuss), the actual thinning process applied to a grid map does not perform this way and, thus, results in an inaccurate GVD. In order to promote the accuracy of the generated GVD, an improved thinning algorithm based on [53] is proposed in this chapter.



(a) (b)

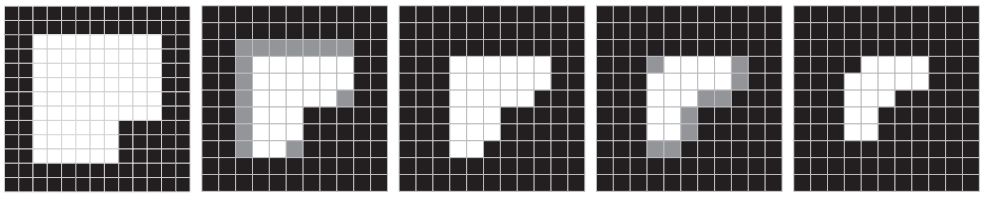
Figure 4.2: a) Ideal thinning process; b) Ideal thinning direction and velocity

4.2 Parallel Thinning Algorithm

Parallel Thinning algorithm [53] is one of the widely used thinning algorithms. It applies a 3×3 pattern (Fig. 4.3) on a grid map, with 0 as obstacles and 1 as unoccupied areas (Setting obstacles as 0 makes it easier to transform a grid map to a clearance map afterwards). Each time based on current cell P1, the algorithm checks the pattern of its eight neighbors to decide whether current cell P1 needs to be set. The parallel thinning algorithm is shown as Alg. 4.1. In each iteration, parallel condition checks run for each cell in the gridmap (lines

P3	P2	P9
P4	P1	P8
P5	P6	P7

Figure 4.3: Thinning pattern



(a) (b) (c) (d) (e)

Figure 4.4: a) Thinning object; b) Mark cells on top-left side of object; c) Set marked cells 0; d) Mark cells on bottom right side of object; e) Set marked cells 0

5 and 11 of Alg. 4.1). Fig. 4.4 shows the thinning process in one iteration. Any cell that meets the conditions will be marked first, and then set as 0. After the first condition check, the cells on the top-left side of the object are marked [line 6 of Alg. 4.1, Fig. 4.4(b)] and set as 0 [line 8 of Alg. 4.1, Fig. 4.4(c)]. After the second condition check, the cells on the bottom-right side of the object are marked [Fig. 4.4(d)] and set as 0 [Fig. 4.4(e)]. As the thinning process continues, there will eventually be only a very thin object.

4.3 Pattern Number

To simplify the algorithm, all patterns are defined with pattern numbers according to the 3×3 neighborhood as proposed in [119]. As shown in Fig. 4.3, each neighbor of grid P1 can be regarded as zero or non-zero. The pattern of the neighborhood can be considered as an eight-bit pattern number, with P2 as the lowest bit and P9 as the highest bit, with 1 as non-zero and 0 as zero (Fig. 4.5).

P9	P8	P7	P6	P5	P4	P3	P2

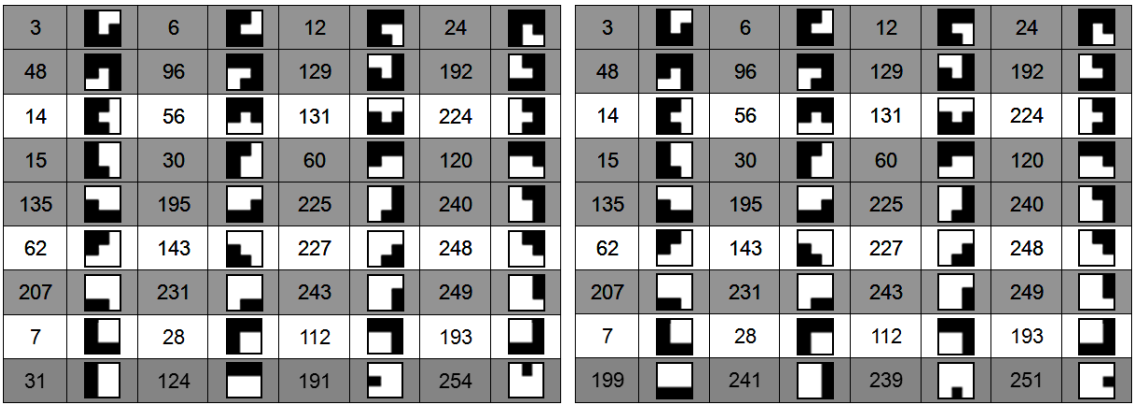
Figure 4.5: Pattern number

In such a manner, all types of cells can be defined with a pattern number between $0 \sim 255$. Based on this number, cells which meet the first condition check in line 5 of Alg. 4.1 are grouped in set Υ_{bottom} [as in Fig. 4.6(b)], and cells that meet the second condition check in

Algorithm 4.1: Parallel thinning algorithm

gridmap: Gridmap, where obstacles are labeled as zero and free space as non-zero
$N(P1)$: Function returns amount of non-zero neighbors of $P1$
$S(P1)$: Function returns number of 0 to 1 (or 1 to 0) transitions in sequence of p2, p3, p4, p5, p6, p7, p8, p9.

```
1  Function ret Thin(gridmap)
2      while while any cell changed do
3          // Mark cells on top-left side of objects.
4          foreach cell P1 in gridmap do
5              if P1 = 1 ∧ 2 ≤ N(P1) ≤ 7 ∧ S(P1) = 1 ∧ P2 · P4 · P8 = 0 ∧ P2 · P6 · P8 = 0
               then
6                  mark P1;
7          // Set marked cells.
8          set all marked cells as 0;
9          // Mark cells on bottom-right side of objects.
10         foreach cell P1 in gridmap do
11             if P1 = 1 ∧ 2 ≤ N(P1) ≤ 7 ∧ S(P1) = 1 ∧ P2 · P4 · P6 = 0 ∧ P4 · P6 · P8 = 0
               then
12                 mark P1;
13         // Set marked cells.
14         set all marked cells as 0;
```



(a) (b)

Figure 4.6: a) Elements of Υ_{top}; b) Elements of Υ_{bottom}

line 11 are grouped in set Υ_{top} [as in Fig. 4.6(a)]. Then the algorithm Alg. 4.1 becomes Alg. 4.2.

Algorithm 4.2: Parallel thinning algorithm with pattern number

$A(P1)$: Function calculating pattern number of P1 (calculated as Fig. 4.5)

```
1  Function ret Thin(gridmap)
2      while while any cell changed do
3          foreach cell P1 in gridmap do
4              if A(P1) ∈ Υ_bottom then
5                  mark P1;
6          set all marked cells as 0;
7          foreach cell P1 in gridmap do
8              if A(P1) ∈ Υ_top then
9                  mark P1;
10         set all marked cells as 0;
```

4.4 Problems of Classic Parallel Thinning Algorithm

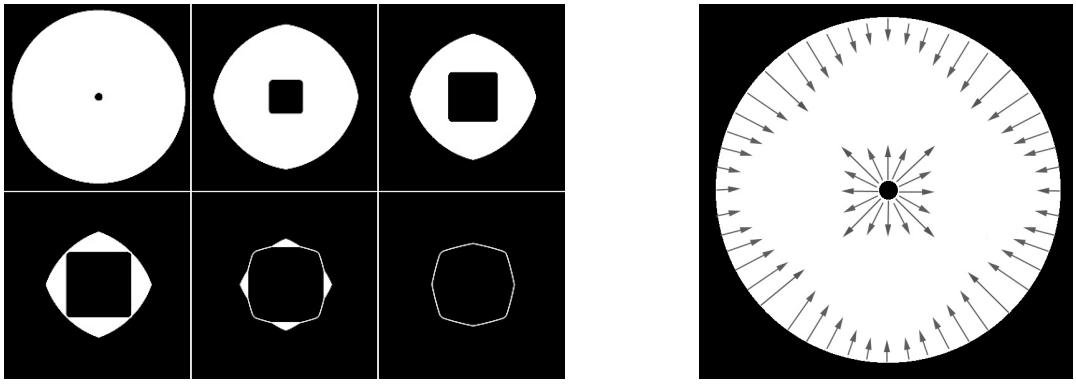
There are two major problems of the classic parallel thinning algorithm, which may result in an inaccurate or ambiguous GVD: first, the square effect that is caused by the special layout of the gridmap, which would produce a very inaccurate GVD; and second, the produced GVD not being thin enough which may lead to ambiguity in the distance evaluation along the GVD.

4.4.1 Square Effect

Compared with the ideal thinning (Fig. 4.2), the square effect of actual classic parallel thinning is shown in Fig. 4.7. It is assumed that the thinning process spreads in all directions with the same speed. However, because classic parallel thinning depends on the 3×3 neighborhood (Fig. 4.3), the layout of the 3×3 neighborhood brings a square effect. In other words, the classic parallel thinning process goes faster in diagonal directions than in vertical and horizontal directions.

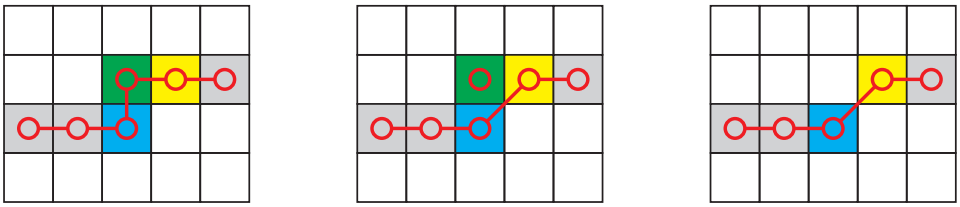
4.4.2 Single Cell-connected GVD

As in Fig. 4.8(a) and 4.8(b), the blue cell connects to a single gray cell on the one side, but on the other side, two cells (yellow and green) connect with the blue cell. This would lead to



(a) (b)

Figure 4.7: Classic parallel thinning process



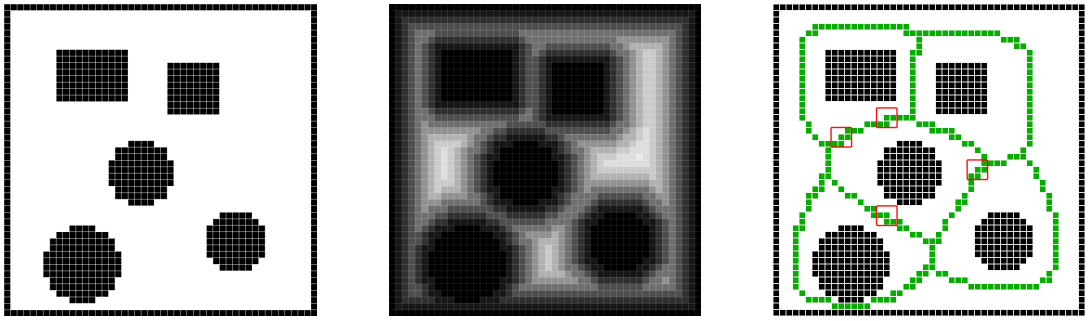
(a) (b) (c)

Figure 4.8: Ambiguity of connecting non-single cell connection

ambiguity, since there are two ways to calculate the length along the GVD as shown in Fig. 4.8(a) and 4.8(b), respectively. Both generate different results. In fact, Fig. 4.8(b) is the correct way to calculate the length. In order to avoid such ambiguity, the green cell needs to be removed [as in Fig. 4.8(c)], so that there is only one way to connect a cell with one of its GVD branches. It can be observed in Fig. 4.9(c) that the green cell is the thinning result from the above sections, and there are some of the non-single cell-connected spots marked. In order to find a more accurate length along the GVD, the diagram needs to be further refined.

4.5 Voronoi-based Parallel Thinning Algorithm

Voronoi-based parallel thinning is proposed to improve the classic parallel thinning algorithm. It solves both abovementioned problems to get an accurate GVD without any ambiguity regarding distance evaluation.



(a) (b) (c)

Figure 4.9: a) Original gridmap; b) Clearance map; c) Thinning result based on clearance map

4.5.1 Clearance-based Thinning

In order to solve the problem of the square effect and produce a more accurate GVD, the thinning algorithm is modified as Alg. 4.3. The grid map is transformed into a clearance map [63], first as Alg. 4.3, line 2 [Fig. 4.9(b)]. Each cell in a clearance map is attached with a clearance value (introduced in Section 2.1.3), which represents its distance to the nearest obstacles [shown as the black cells in Fig. 4.9(a)]. The darker the cell, the nearer it is to the obstacles. The thinning process starts with the cell of the lowest clearance value (Alg. 4.3, line 4), then iteratively goes on to cells with larger clearance values (Alg. 4.3, line 5). The thinning process is sequentially applied to cells with different clearance values until all cells have been processed. With the help of the clearance map, the square effect can be avoided [as in Fig. 4.9(c)].

4.5.2 Refinement

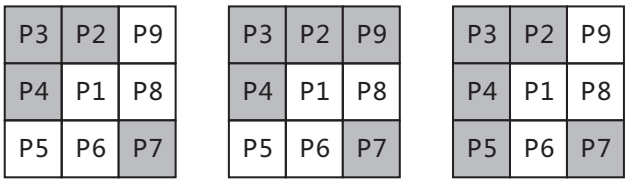


Figure 4.10: Types of non-single cell-connected pattern

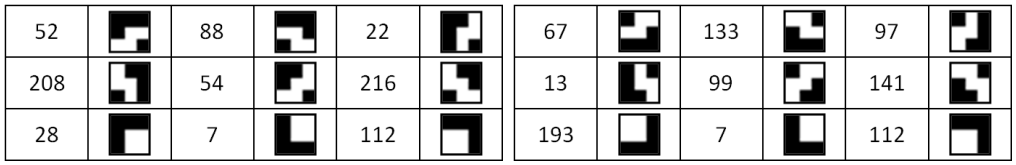
Several types of non-single cell-connected spots are marked with a red square [as in Fig. 4.9(c)]. There are three types of non-single cell-connected patterns (as in Fig. 4.10). These patterns and their variations are grouped and integrated as supplementary elements in Υ_{top} and Υ_{bottom}, respectively, as shown in Fig. 4.11(a) and 4.11(b). The solution for the second problem is to include the supplementary elements into Υ_{top} and Υ_{bottom}. After

Algorithm 4.3: Voronoi-based parallel thinning algorithm

$A(P1)$: eight-bit integer (calculated as Fig. 4.5)

```
1  Function ret Thin(gridmap)
2      clearmap ← distance_transform(gridmap);
3      max_clear ← get the maximum clearance value in clearmap;
4      d ← 1.5;
5      while d ≤ max_clear do
6          foreach cell P1 in gridmap, clear[P1] < d do
7              if A(P1) ∈ Υ_bottom then
8                  mark P1;
9          set all marked cells as 0;
10         foreach cell P1 in gridmap, clear[P1] < d do
11             if A(P1) ∈ Υ_top then
12                 mark P1;
13         set all marked cells as 0;
14         d ← d + 1;
```

the abovementioned steps, all non-single cell-connected spots that can create ambiguity in calculating the length will disappear, and an accurate GVD can be extracted from the grid map (as Section 7.1.2 will show).



(a) (b)

Figure 4.11: a) Supplementary elements of Υ_{top}, b)Supplementary elements of Υ_{bottom}

4.6 Summary

A Voronoi-based parallel thinning algorithm (VPT) is proposed to extract the GVD based on the grid map. Since the classic parallel thinning algorithm always comes with a square effect, it will severely affect the accuracy of the GVD. The proposed VPT algorithm could avoid such a square effect and the generated GVD is always single cell-connected, which makes it easier to evaluate the distance along the GVD. Compared to the polygon-based

(i.e. involving lines and curves) algorithms, the VPT algorithm can be directly applied to the grid map. The thinning is done only on the 3×3 neighbors of the current cell and the VPT processes the cells one by one so that the computation time only relies on the size of the map instead of the complexity. This makes the VPT a great tool for dealing with the clustered environment.

Chapter 5

Global Path Planning

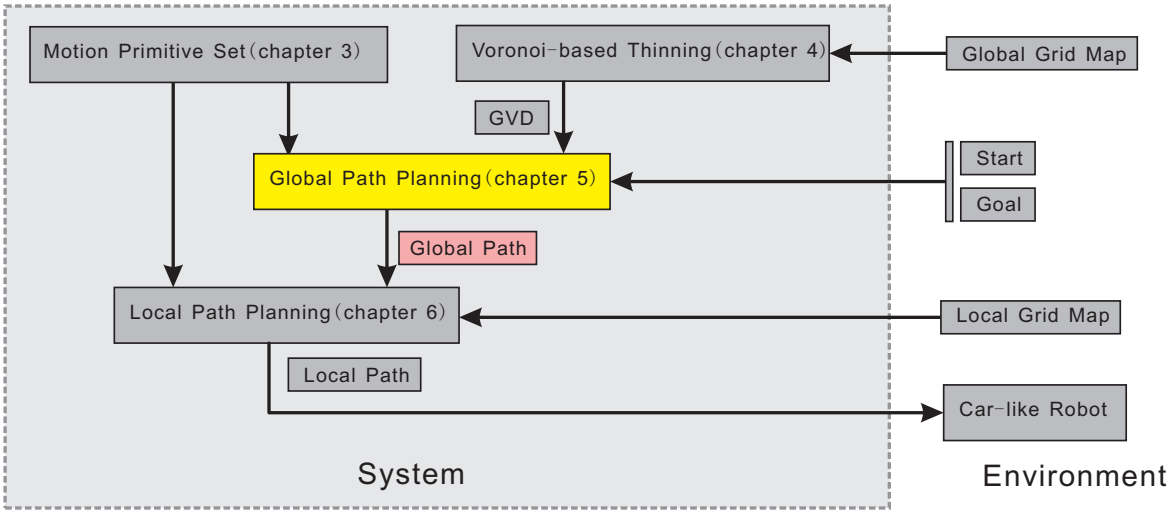


Figure 5.1: New Voronoi-based path planning process: global path planning

[1]The highlighted yellow box in Fig. 5.1 shows the location of global path planning in the system. The search firstly forms a global path with the primitive trajectory set produced in Chapter 3. As mentioned in Section 1.2, the local minimum is the major problem slowing down search-based path planning. The local minimum results from the application of the Euclidean-based heuristic. Although a number of solutions [11–16, 90–92] have been developed to reduce the computation time of search-based path planning, the topic of heuristic itself has barely been touched. A new type of heuristic, measured with the help of the GVD, is proposed in this chapter. The proposed heuristic provides a more accurate estimate of the cost to the goal and, therefore, is able to navigate the search to avoid the local minimum, dramatically speeding up the global path planning process.

[1]Part of the work in this chapter has been published at [120]

5.1 Ideal Heuristic

It costs a large amount of memory and computation time to universally search the space (as discussed in Section 2.2.2). In order to make the search more goal-oriented, the A* algorithm integrates a heuristic cost into the estimation of each node, which guides the direction of the exploring tree.

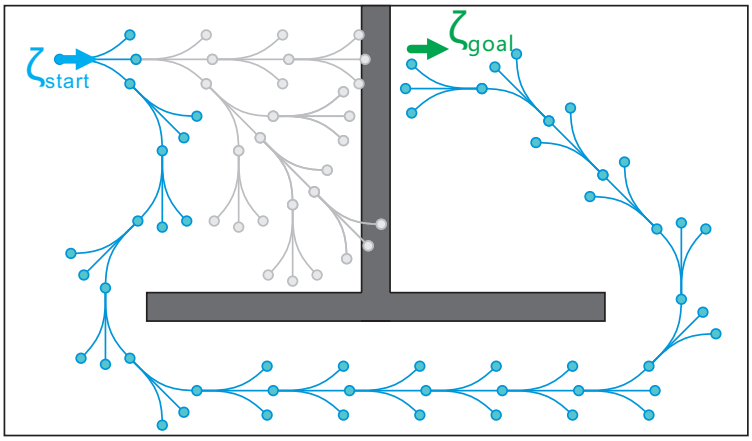


Figure 5.2: Exploring tree with ideal heuristic cost

As mentioned in Chapter 2, the Euclidean distance is still the most widely used heuristic. Fig. 5.2 shows the problem of the Euclidean-based heuristic. The gray trajectories show the behavior of the search applying the Euclidean-based heuristic. As mentioned in Section 2.2.2, the exploring tree always grows towards the goal. The blue trajectories represent the ideal searching direction, but the search will not go for that until the concave area has been totally searched. Fully searching the concave area costs extra computation time and memory cost. When the robot moves in a large, clustered area that consists of many concave objects, it may either run out of memory or take too long to get a result. For large-scale, clustered environments, it may take a very long time until the search eventually gets rid of all the local minima, making the search with the Euclidean-based heuristic a very time-consuming process.

5.2 Voronoi-based Heuristic

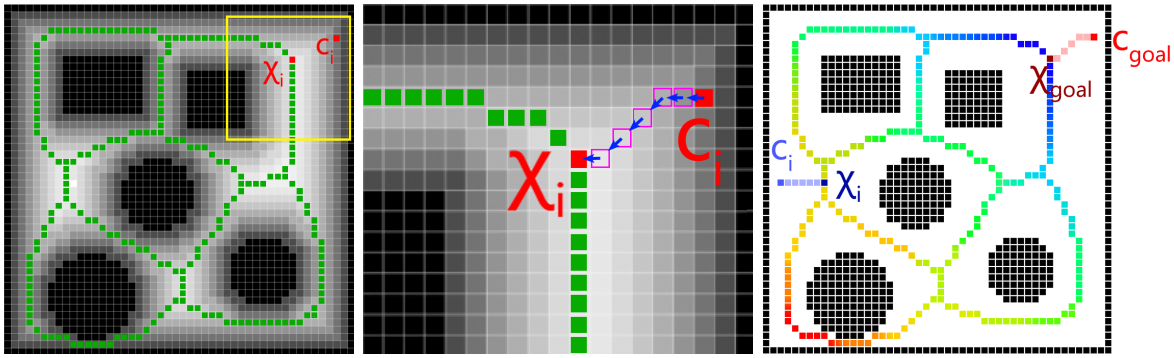
The GVD can be understood as a collection of medial axes between the obstacles which forms a roadmap that accurately describes the geometry of the environment (as in Chapter 4). For each state ζ_i in the free space, position χ_i on the GVD, which is closest to ζ_i, is defined as the corresponding GVD position of ζ_i. The GVD position provides the connection from any state in the space to the GVD. The localization of the GVD position for a certain state ζ_i is provided in next Section (5.2.1). The Voronoi-based heuristic estimates the time

cost between arbitrary state ζ_i and goal ζ_{goal} by first determining position χ_{goal} on the GVD closest to goal ζ_{goal} and position χ_i closest to currently targeted state ζ_i. Consecutively, the algorithm determines the minimum cost between χ_{goal} and χ_i on the GVD by applying the Voronoi cost distribution (explained later in Section 5.2.2).

The Voronoi-based heuristic is the estimated cost evaluated based on the cost along the GVD, which provides a more accurate measurement compared to the Euclidean distance. The proposed Voronoi-based heuristic is not simply a distance cost along the GVD, but an estimated time cost that the robot may spend on the way.

5.2.1 GVD Position

As introduced in Section 3.1, the configuration of the car-like robot is represented with $\zeta_i(x_i, y_i, \theta_i)$, where x_i and y_i represent the position of the robot and θ_i is the direction of the robot. For simplicity, here the configuration $\zeta_i(x_i, y_i, \theta_i)$ of the robot is reformed as $\zeta_i(c_i, \theta_i)$, where $c_i(x_i, y_i)$ is used to represent the position of ζ_i.



(a) (b) (c)

Figure 5.3: a) Clearance map overlaid by GVD; b) Subarea around position c_i of ζ_i; c) Cost distribution along GVD

Fig. 5.3(a) is the clearance map overlaid by the GVD. For any given configuration $\zeta_i(c_i, \theta_i)$, there is always a corresponding position χ_i on GVD, χ_i is defined as the GVD position of ζ_i. The GVD position can be found with the help of the clearance map by iteratively moving from position c_i of ζ_i along the direction of the gradient on the clearance map [shown as arrows in Fig. 5.3(b)].

5.2.2 Time Cost Distribution over GVD

As shown in Fig. 5.3(c), for given goal configuration ζ_{goal} in the free space, its GVD position can be found as χ_{goal}. Based on χ_{goal}, any cell on the GVD will be attached with a cost representing the cost from the current position to χ_{goal} along the GVD. This is done with the

Dijkstra's algorithm [88] as Alg. 5.1. All costs of GVD cells are initiated as infinite, except the cost of χ_{goal} that is set as 0 (Alg. 5.1, line 2$\sim$ 6). The cost is then distributed from χ_{goal} over the GVD (Alg. 5.1, line 7$\sim$16), where $cost\_between(\chi_c, \chi_b)$ calculates the cost between χ_c and χ_b (Alg. 5.1, line 12). The definition of $cost\_between(\chi_c, \chi_b)$ is presented in the next section. As shown in Fig. 5.3(c), the transition of the color along the GVD shows how the cost is distributed on it; the blue sections are close to the goal and the red sections are far from the goal. Since the GVD is the highly extracted geometry of the environment, the cost distribution is generated very fast even for a large-scale environment. From the cost distribution, any configuration ζ_i in the free space can now be evaluated based on $cost[\chi_i]$ as shown in Fig. 5.3(c), where χ_i is the GVD position of ζ_i.

Algorithm 5.1: Cost distribution

```
1  Function CostDistribution()
2      foreach χ_i ∈ X_GVD do
3          cost[χ_i] ← infinity;
4      cost[χ_goal] ← 0;
5      openset ← ∅ ;
6      openset ← openset ∪ {χ_goal};
7      while openset ≠ ∅ do
8          χ_c ← the GVD position with the minimum cost in openset;
9          openset ← openset − {χ_c};
10         foreach neighbor GVD position χ_b of χ_c do
11             // cost_between(χ_c, χ_b) calculates cost between χ_c and χ_b, which
                  is proposed in Section 5.2.3.
12             new_cost ← cost[χ_c] + cost_between(χ_c, χ_b);
13             if new_cost < cost[χ_b] then
14                 cost[χ_b] ← new_cost;
15                 if χ_b ∉ openset then
16                     openset ← openset ∪ {χ_b};
```

5.2.3 Cost Function of Neighboring GVD Positions

Instead of the distance cost, the cost function between the neighboring cells is actually the time cost that the robot moves between them. As in (5.1), $dist(\chi_a, \chi_b)$ is the distance between χ_a and χ_b. $\overline{v}(\chi_a, \chi_b)$ is the velocity, with which the robot moves from χ_a to χ_b. As (5.2), $\overline{v}(\chi_a, \chi_b)$ is the average of the estimated velocity on χ_a and χ_b. The estimated velocity on χ_a and χ_b are defined by (5.3) and (5.4). However, (5.3) and (5.4) are defined in a general

form for any given position c_i, which can also be GVD positions.

$$cost\_between(\chi_a, \chi_b) = \frac{dist(\chi_a, \chi_b)}{\overline{v}(\chi_a, \chi_b)} \tag{5.1}$$

$$\overline{v}(\chi_a, \chi_b) = \frac{v(\chi_a) + v(\chi_b)}{2} \tag{5.2}$$

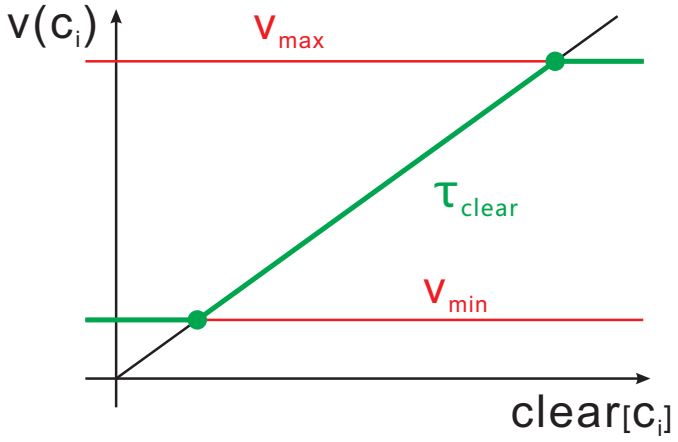


Figure 5.4: Estimated velocity propositional to clearance

Obviously, the position c_i with larger clearance value $clear[c_i]$ is relatively far from the obstacles, and provides higher safety and a lower possibility of traffic jams. Therefore, the estimated velocity of the robot on c_i is allowed to be higher than that on the positions with lower clearance values. The estimated velocity of the mobile robot at c_i (as in Fig. 5.4) is defined proportionally to its clearance value $clear[c_i]$ as (5.4), where τ_{clear} is the coefficient to transform the effect of clearance into velocity. However, since the movement of the mobile robot is also limited by its maximum velocity v_{max} and minimum velocity v_{min}, $v(c_i)$ can only vary between v_{max} and v_{min}.

$$v(c_i) = \begin{cases} v_{max}, & v_{clear}(c_i) \geq v_{max} \\ v_{clear}(c_i), & v_{max} > v_{clear}(c_i) \geq v_{min} \\ v_{min}, & v_{min} > v_{clear}(c_i) \end{cases} \tag{5.3}$$

$$v_{clear}(c_i) = \tau_{clear} clear[c_i] \tag{5.4}$$

Based on such a relationship, higher clearances will produce higher velocities and, in turn, will lead to lower time costs. This will help distribute the time cost faster along the GVD sections with higher clearances (in other words, the corridor sections that are wider).

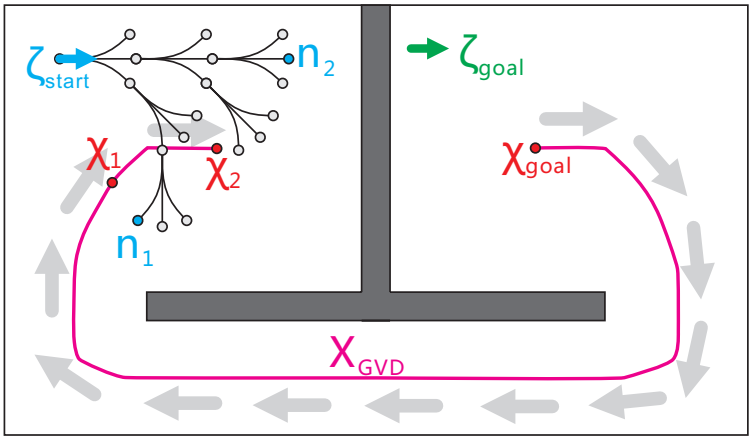


Figure 5.5: Evaluating nodes with GVD

5.2.4 GVD-based Heuristic Estimate

As shown in Fig. 5.5, the heuristic estimate measures the cost along the GVD. The pink line is the GVD created with the VPT algorithm (Chapter 4). X_{GVD} represents the collection of all GVD positions. χ_{goal} is the GVD position of ζ_{goal}. n_1 and n_2 are two nodes in the exploring tree with GVD positions χ_1 and χ_2 , respectively. As shown in Fig. 5.5, the cost is distributed along the GVD based on χ_{goal} (shown as the arrows). Each cell on the GVD is set with a value based on the cost function (5.1). The heuristic estimations of n_1 and n_2 are $cost[\chi_1]$ and $cost[\chi_2]$, respectively. Clearly, χ_1 is nearer to χ_{goal} along the GVD than χ_2, so that $h(n_1) < h(n_2)$. Based on (2.1), this yields $f(n_1) < f(n_2)$ according to Fig. 5.5. This means node n_1 will be explored in the next iteration which will eventually produce the result shown in Fig 5.6. With the help of the GVD, the exploring tree can efficiently avoid being trapped in the local minimum.

5.2.5 Goal Zone

However, this approach can only guide the search to χ_{goal} instead of the real configuration of goal ζ_{goal}. To let the search eventually reach ζ_{goal}, a goal zone is defined as shown in Fig. 5.6. The goal zone is a set of GVD positions X_{goal} (shown as the green section of GVD in Fig. 5.6). X_{goal} is defined by (5.5), where $dist(\chi_i, \chi_{goal})$ is the distance between χ_i and χ_{goal}, $clear[\chi_{goal}]$ is the clearance value of χ_{goal}, and X_{GVD} is the collection of all GVD positions.

$$X_{goal} = \{\chi_i \mid dist(\chi_i, \chi_{goal}) \leq clear[\chi_{goal}], \chi_i \in X_{GVD}\} \tag{5.5}$$

Any node n_i with the GVD position of $\chi_i \in X_{goal}$ is considered in the goal zone. As shown in Fig. 5.6, the GVD position χ_2 of n_2 is inside the goal zone, since $dist(\chi_2, \chi_{goal}) < clear[\chi_{goal}]$, and $\chi_2 \in X_{goal}$. However, the GVD position χ_1 of n_1 does not belong to X_{goal}, so n_1 is not in the goal zone.

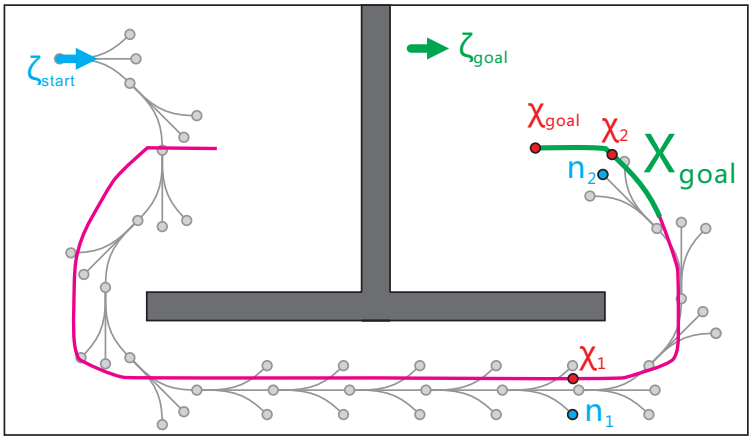


Figure 5.6: Goal zone X_{GVD}

5.2.6 Heuristic Function

The Voronoi-based heuristic is defined by Alg. 5.2. It provides an estimated time cost between current node n_i and goal ζ_{goal}.

As in Alg. 5.2, line 3, when $\chi_i \in X_{goal}$, (χ_i is the GVD position of n_i), it means n_i and ζ_{goal} are in the same convex area and there are no obstacles between them (as n_2 in Fig 5.6). Therefore, the heuristic cost of n_i can be calculated based on (5.6), which is the time cost between n_i and ζ_{goal} based on the generalized Euclidean distance between ζ_i and ζ_{goal}, where ζ_i is the state of n_i.

$\| \zeta_i - \zeta_{goal} \|$ in (5.6) is the generalized Euclidean distance between ζ_i and ζ_{goal}, defined by (5.7) in 3D space. κ_θ in (5.7) is the coefficient that transfers the angular difference between ζ_i and ζ_{goal} into distance.

$\overline{v}(c_i, c_{goal})$ in (5.6) is the estimated average velocity of robot moving from ζ_i to ζ_{goal}, and is calculated in the same way with (5.2), where c_i and c_{goal} are the position of ζ_i and ζ_{goal}, respectively.

$$EuclideanHeuristic(n_i) = \frac{\| \zeta_i - \zeta_{goal} \|}{\overline{v}(c_i, c_{goal})} \tag{5.6}$$

$$\| \zeta_i - \zeta_{goal} \| = \sqrt{(x_i - x_{goal})^2 + (y_i - y_{goal})^2 + \kappa_\theta(\theta_i - \theta_{goal})^2} \tag{5.7}$$

As in Alg. 5.2, line 8, when $\chi_i \notin X_{goal}$, which means n_i and the goal ζ_{goal} may be not in the same convex area (as n_1 in Fig. 5.6). As a result, the time cost between n_i and ζ_{goal} must be evaluated based on the time cost distribution $cost[\chi_i]$ over the GVD in Section 5.2.2, where χ_i is the GVD position of n_i. However, $cost[\chi_i]$ only provides the time cost from χ_i to χ_{goal}, rather than the time cost from χ_i to ζ_{goal} (χ_{goal} is the GVD position of goal ζ_{goal}).

Algorithm 5.2: Voronoi-based heuristic

n_i : Input node
ret: Voronoi heuristic of n_i

```
1 Function ret VoronoiHeuristic(n_i)
2     χ_i ← the GVD position of n_i;
3     if χ_i ∈ X_goal then
4         // When n_i is in goal zone.
5         return EuclideanHeuristic(n_i);
6     else
7         // When n_i is not in goal zone.
8         return cost[χ_i] + remain_cost;
```

Therefore, apart from $cost[\chi_i]$, the time cost between χ_{goal} to ζ_{goal} should also be added, which is represented by $remain\_cost$ in Alg. 5.2, line 8.

The time cost between χ_{goal} to ζ_{goal} ($remain\_cost$) is also calculated based on the generalized Euclidean distance between χ_{goal} and ζ_{goal}, which is in the same way with (5.6). However, since χ_{goal} is only a 2D position, whereas ζ_{goal} is a 3D state, the generalized Euclidean distance can not be directly calculated between χ_{goal} and ζ_{goal}. Therefore, a 3D auxiliary state ζ'_{goal} is created at the same position of χ_{goal} as shown in Fig. 5.7. The direction of ζ'_{goal} is pointed from χ_{goal} to the position of ζ_{goal}. As shown in Fig. 5.7, ζ'_{goal} can be regarded as the estimated orientation to approach the goal ζ_{goal} from χ_{goal}.

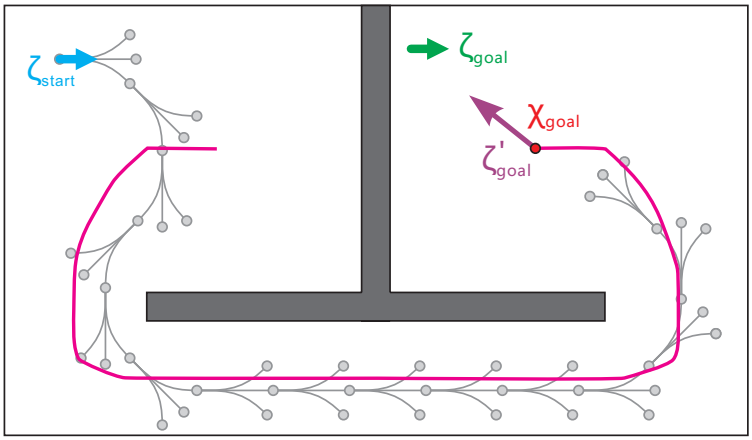


Figure 5.7: Auxiliary state ζ'_{goal} is used to calculated generalized Euclidean distance between χ_{goal} and ζ_{goal}

As a result, the time cost between χ_{goal} and ζ_{goal} can be calculated with the help of auxiliary state ζ'_{goal} as (5.8). c'_{goal} is the position of ζ'_{goal}. There is $c'_{goal} = \chi_{goal}$, because ζ'_{goal}

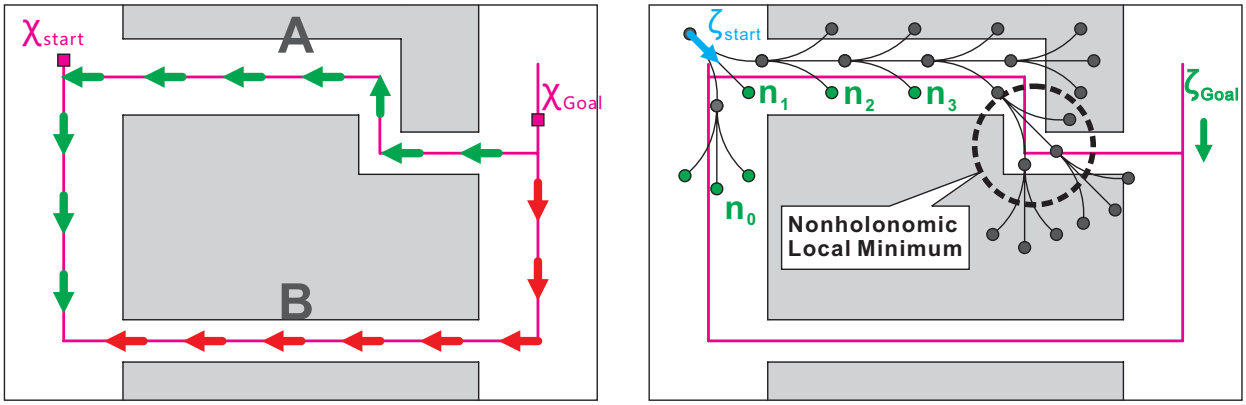
is created on the same position of χ_{goal},

$$remain\_cost = \frac{\| \zeta'_{goal} - \zeta_{goal} \|}{\overline{v}(c'_{goal}, c_{goal})} \tag{5.8}$$

5.3 Nonholonomic Local Minimum Avoidance

With the help of the Voronoi-based heuristic, the search can avoid the local minimum caused by the obstacles. However, making use of the GVD could lead to another type of local minimum, which is caused by the combination of the obstacles and the nonholonomic constraints.

5.3.1 Nonholonomic Local Minimum



(a) (b)

Figure 5.8: a) Directions of cost distribution; b) Nonholonomic local minimum

As shown in Fig. 5.8, there are corridors A and B, both of which can lead to the goal. The pink lines located in the center of the corridor are the GVD of the environment. Clearly, the path through corridor A is shorter than through corridor B. This can also be observed from the cost distribution in Fig. 5.8(a). The red and green arrows show the cost distribution along different GVD sections. Based on the cost distribution, the search is led to corridor A first. However, corridor A includes a sharp corner that violates the nonholonomic constraints of the car-like robot. This makes corridor A untraversable, which can be seen in Fig. 5.8(b). As mentioned, the search is based on the A* algorithm. The dark nodes are the nodes in the closed set which either are fully explored or have a collision against the obstacle. Green nodes $n_0 \sim n_3$ are the ones still in the open set. According to the cost distribution along the GVD in Fig. 5.8(a), it is not hard to infer that the Voronoi-based heuristic estimation of nodes $n_1 \sim n_3$ would be less than the estimate of n_0. This makes nodes $n_1 \sim n_3$ be explored

first before n_0. Therefore, the search will not explore corridor B until corridor A is fully explored, which results in the accumulated nodes around the corner and, consequently forms another type of local minimum. Since such a type of local minimum is caused by the obstacles and the nonholonomic constraints, it is defined as the nonholonomic local minimum (NLM) marked with a dashed circle in Fig. 5.8(b) to be differentiated from the local minimum that is caused by obstacle constraints only.

5.3.2 Nonholonomic Local Minimum Detection

In order to deal with the problem of the NLM, as long as there is a new node created, the following function of Alg. 5.3 will be called for this node. At line 4 of Alg. 5.3, r_{nlm} defines the size of an effective area of χ_i on the GVD, where χ_i is the GVD position of n_i. For any GVD positions χ_j that locates inside the effective area of χ_i, i.e. $dist(\chi_j, \chi_i) < r_{nlm}$, the clearance value of χ_j will be reduced slightly by $clear_{nlm}$ (as at line 6), where $clear_{nlm}$ defines the size of the minimum space that each node occupies.

Algorithm 5.3: Nonholonomic local minimum detection

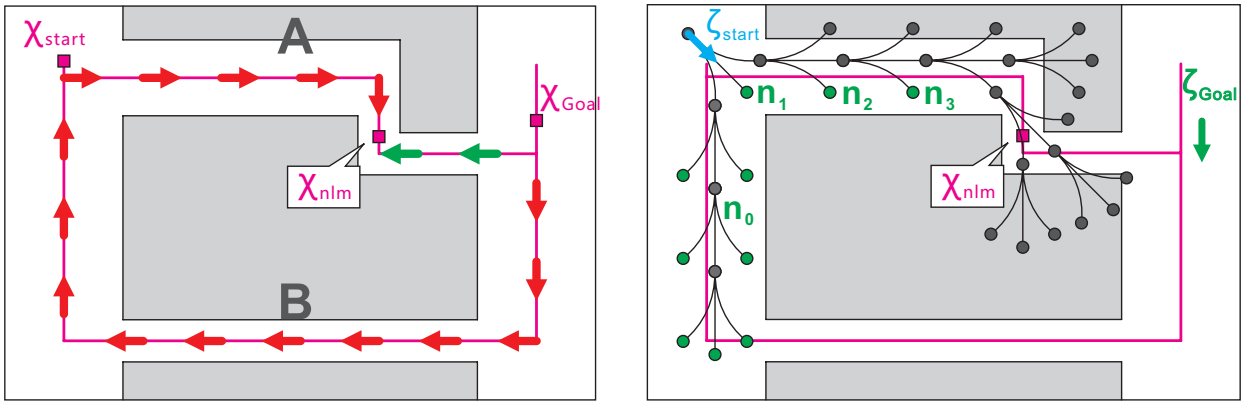
n_i: The input node

```
1  Function ret NLM_Detection(n_i)
2      χ_i ← GVD position of n_i;
3      // For each state χ_j in the small neighborhood of χ_i.
4      foreach GVD position χ_j, dis(χ_j, χ_i) < r_nlm do
5          // Reduce clearance value of χ_j slightly by clear_nlm.
6          clear[χ_j] ←  clear[χ_j] − clear_nlm;
7          if clear[χ_j] < 0 then
8              // A NLM is detected.
9              return true;
10     return false;
```

When the search reached an NLM, it will be blocked by the NLM. Therefore, the nodes will quickly accumulate around χ_i. This makes the clearance values of the GVD section around the NLM dramatically decrease. As the nodes keep accumulating, the clearance value of a certain GVD position χ_j around the NLM (line 7 in Alg. 5.3) will eventually fall below zero. Then, χ_j is detected as an NLM.

5.3.3 Update Cost Distribution

Alg. 5.4 is the modified A* algorithm with NLM avoidance. Alg. 5.4 is basically the same with the original A* algorithm Alg. 2.4, but with some modification at line 23 and lines 27∼32. At line 23, the heuristic cost is replaced with the Voronoi-based heuristic. The NLM



(a) (b)

Figure 5.9: a) Reproduced cost distribution; b) Based on regenerated cost distribution, exploring tree grows away from NLM.

detection and avoidance are added between line 27 and line 32. As long as an NLM (χ_{nlm}) is found on the GVD (line 27), the cost distribution over GVD will be regenerated (line 29). As mentioned in Section 5.3.2, the clearance value dramatically decreases around χ_{nlm}, so that $clear[\chi_{nlm}] < 0$.

Based on the definition of (5.1) in Section 5.2.3, a low clearance value will result in a high time cost. Therefore, the cost over χ_{nlm} could be very high and this will change the direction of the cost distribution over the GVD. As Fig. 5.9(a) shows, the high cost over χ_{nlm} makes the cost distribution along the green arrows stop at χ_{nlm}, so the cost has to be distributed from another direction along the red arrows. All nodes in *openset* of the exploring tree will be then updated with the new heuristic value (line 32). Now the problem of the NLM is solved, as shown in Fig. 5.8(b). According to the new updated cost distribution, the heuristic estimate of n_0 becomes lower than the estimates of $n_1 \sim n_6$, so n_0 will be chosen as the current node to be explored and the exploring tree will follow the updated cost distribution through corridor B without fully searching corridor A. The NLM can now be efficiently avoided (as the experiment shows later in Section 7.2.2), saving time and memory cost.

5.4 Summary

In this chapter, the Voronoi-based heuristic is proposed to deal with the local minimum problem caused by the obstacles and the nonholonomic constraints of the car-like mobile robot. The Voronoi-based heuristic provides a more accurate estimate to the goal so that the search can be guided away from the local minimum. This largely saves computation time during the search. Rather than the distance cost, the time cost is used to estimate the

cost to the goal, which is inversely proportional to the clearance along the GVD sections. Therefore, the GVD sections of the narrow corridors with a lower clearance value will be attached with larger time costs. This helps the robot avoid narrow corridors and, thus, minimize the possibilities of traffic jams. Since the GVD is the highly extracted geometrical information of the map, the Voronoi-based heuristic can efficiently avoid the obstacle-based local minimum. Then, the nonholonomic local minimum is introduced, which is caused by the obstacles and the nonholonomic constraints. A corresponding solution is also proposed to detect the nonholonomic local minimum and update the cost distribution over the GVD during the search.

Algorithm 5.4: Nonholonomic local minimum avoidance-based A*

```
1  Function ret A_Star(ζ_start, ζ_goal)
2      n_s.ζ ← ζ_start;
3      n_s.parent_node ← null;
4      initiate g_score[] of all states infinite;
5      g_score[n_s.ζ] ← 0;
6      n_s.f ← g_score[n_s.ζ] + heuristic(n_s.ζ, ζ_goal);
7      openset ← {n_s};
8      closedset ← ∅;
9      while openset ≠ ∅ do
10         n_c ← node in openset with lowest f cost;
11         if ζ_c = ζ_goal then
12             return reconstruct_path(n_c);
13         openset ← openset − {n_c};
14         closedset ← closedset ∪ {n_c};
15         foreach node n_i ∈ neighbor_nodes(n_c) do
16             if n_i ∈ closedset then
17                 continue;
18             // path_cost(n_c, n_i) calculates cost between n_c and n_i, which is
                   defined in Section 3.3.
19             new_g_score ← g_score[n_c.ζ] + path_cost(n_c, n_i);
20             if n_i ∉ openset or new_g_score < g_score[n_i.ζ] then
21                 n_i.parent_node ← n_c;
22                 g_score[n_i.ζ] ← new_g_score;
23                 n_i.f ← new_g_score + VoronoiHeuristic(n_i);          // Alg. 5.2
24                 if n_i ∉ openset then
25                     openset ← openset ∪ {n_i};
26                 // NLM avoidance.
27                 if NLM_Detection(n_i) = true then                     // Alg. 5.3
28                     // Regenerate cost distribution with Alg. 5.1.
29                     CostDistribution();
30                     // Update cost of each node in openset.
31                     foreach node n_i ∈ openset do
32                         n_i.f ← g_score[n_i.ζ] + VoronoiHeuristic(n_i);
33     return failure;
```

Chapter 6

Local Path Planning

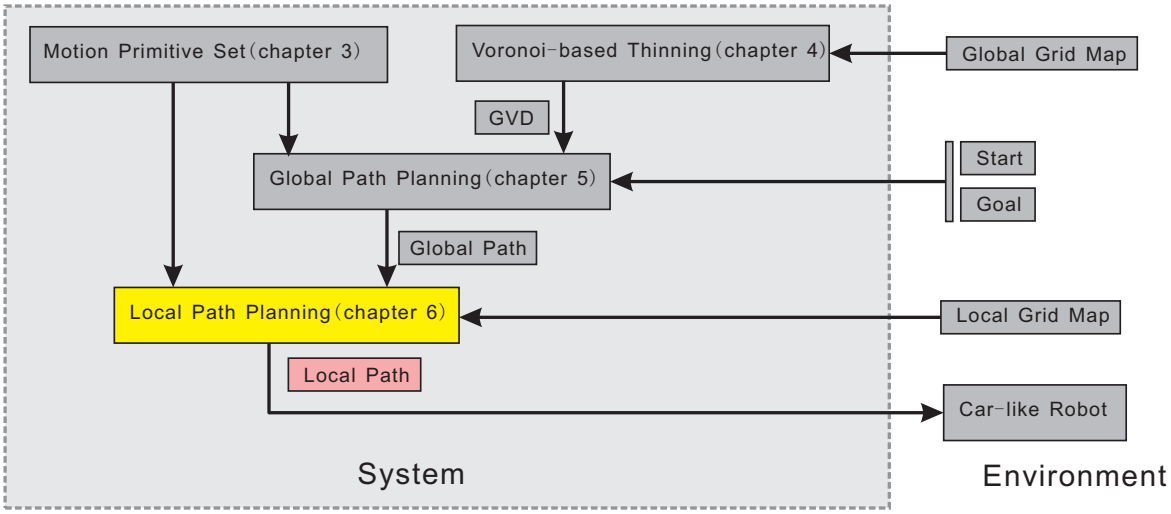


Figure 6.1: New Voronoi-based path planning process: local path planning

[1]This chapter proposes a new corridor-based local path planner. Local path planning serves as a supplement to global path planning to avoid the dynamic obstacles in the environment while still following the guidance of the global path. As the highlighted yellow box in Fig. 6.1 shows, the local path planning process uses the same primitive trajectory set as global path planning. Instead of following any specific global path as mentioned in Section 1.2, the proposed local path planner is guided by normative path-based heuristic (proposed in Section 6.1.5) through a sequence of corridors [122] (Section 6.1.2) through which the global path also passes (as in Fig. 6.1). As evident in the pink box in Fig. 6.1, the local planner generates a local path that navigates the robot through the corridors and approaches the goal.

[1]Part of the work in this chapter has been published at [121]

As mentioned in Section 2.5, the existing solution [22,23,113–116] are generally applying simultaneous path following and obstacle avoidance (SPFOA). SPFOA tries to follow the global path as much as possible, and makes only minor modifications to avoid dynamic obstacles. This increases the possibility of the local minimum if a concave obstacle happens to be on the global path. The proposed corridor-based local path planner is not required to precisely follow the global path. This lends the local planner more flexibility to avoid the dynamic obstacles on the way and reduces its possibility of being trapped by the local minimum.

The proposed corridor-based local path planner requires the search to be done in a path corridor (Section 6.1.2), rather than the entire search space. The path corridor [122] defines a small subspace around the global path. It largely reduces the computation time, ensuring that the generated path will neither be too far away from the global path nor lose the flexibility of avoiding dynamic obstacles.

6.1 Path Corridor-based Local Path Planner

With the global planner proposed in previous chapters, a traversable global path can be found with the minimized time cost. The time cost is affected by steering rate, clearance, distance, etc. The global path could form extra constraints that require the robot always to follow, seriously disturbing the flexibility of local planning. Given this problem, global path P_{global} will be transformed into normative path P_{norm} first, instead of being applied directly.

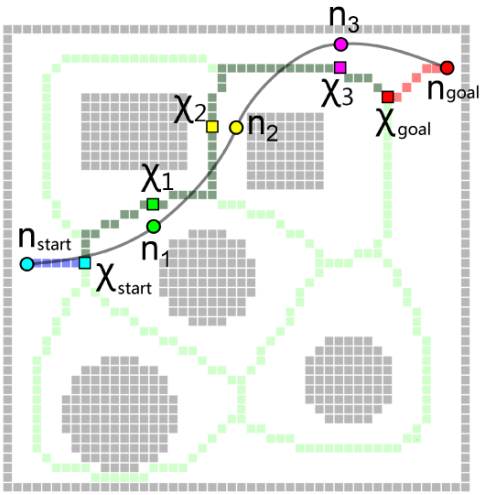


Figure 6.2: Normative path exaction

6.1.1 Normative Path Extraction

At first, normative path P_{norm} is initiated as the connection of start point n_{start} to GVD position χ_{start} (shown as blue cells in Fig. 6.2). Then, for each path section between n_i and n_{i+1} of the global path, the GVD section between χ_i and χ_{i+1} will be appended to P_{norm} (shown as green cells in Fig. 6.2), where χ_i and χ_{i+1} are the GVD positions of n_i and n_{i+1}, respectively. Finally, the connection of the goal to its GVD position χ_{goal} is added to the end of P_{norm} (shown as red cells in Fig. 6.2).

6.1.2 Path Corridor

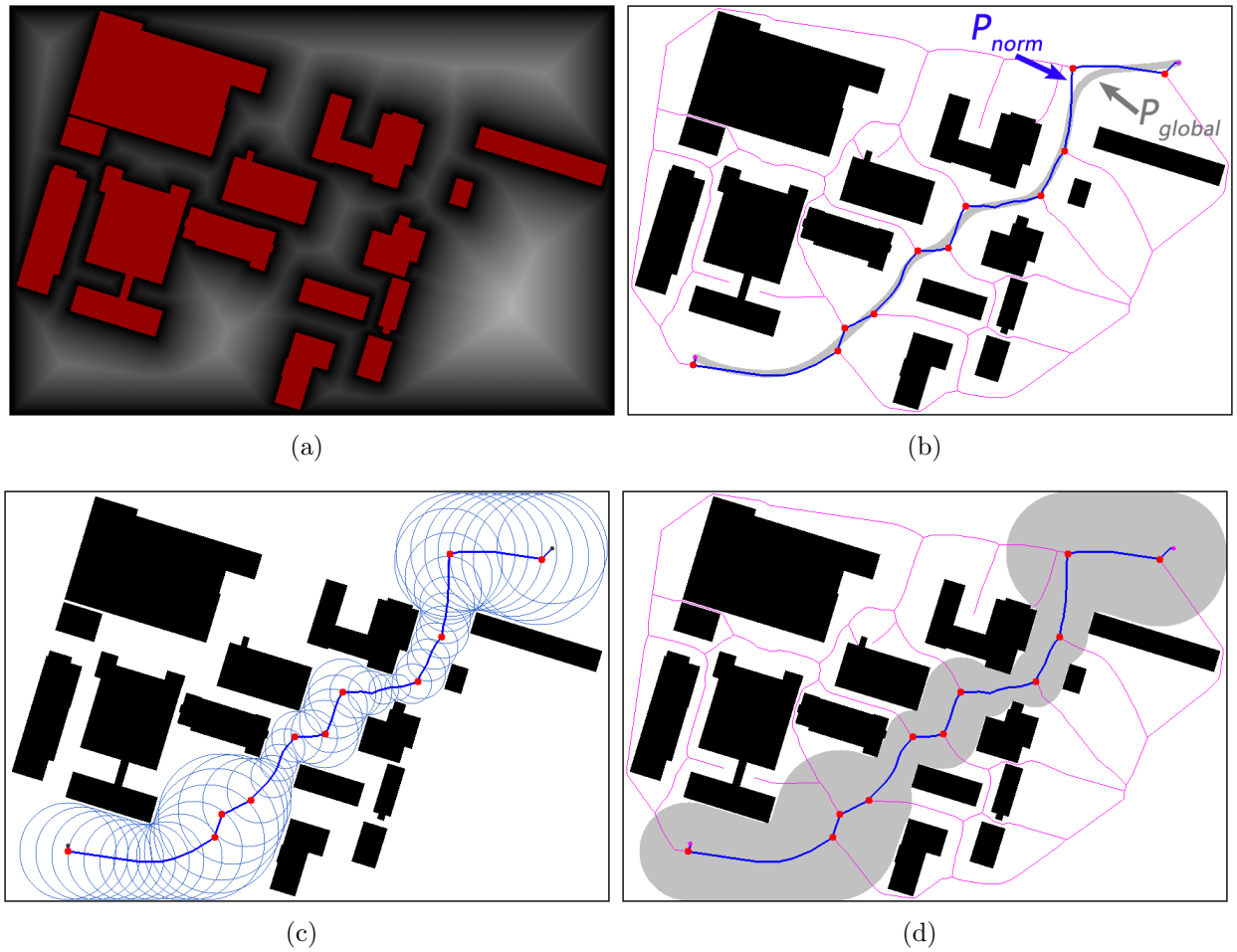


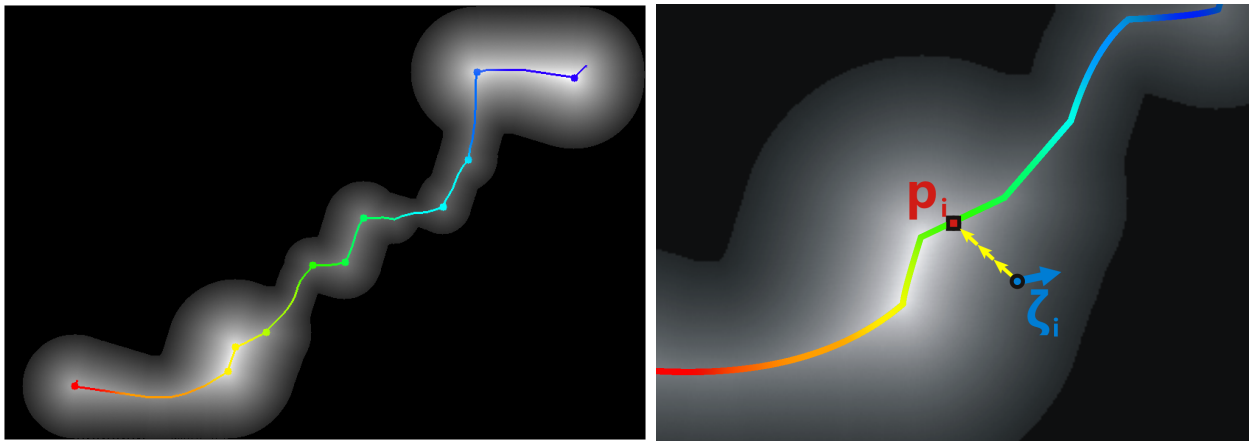
Figure 6.3: a) Clearance map; b) GVD topological map; c) GVD overlaid by global path; d) Generation of path corridor

Fig. 6.3(a) is the clearance map and Fig. 6.3(b) is the GVD overlaid by global path P_{global} (the gray line) and normative path P_{norm} (the blue line). As shown in Fig. 6.3(c), there is

a sequence of disks centered on each path position $p_i \in P_{norm}$, with the radius of $clear[p_i]$. $clear[p_i]$ is the value of p_i in the clearance map as shown in Fig 6.3(a). This produces a path corridor [122] as shown in Fig. 6.3(d). The path corridor could be considered as the maximum space along P_{norm}, so P_{norm} naturally becomes the medial line of the path corridor. For local path planning, the area outside the corridor becomes irrelevant and only the area inside the corridor is considered as free space [the gray area in Fig. 6.3(d)].

6.1.3 Corridor Clearance Map, Path Position and Normative Path Cost

Based on the path corridor and normative path P_{norm}, corridor clearance map $path\_clear[]$ [Fig. 6.4(a)] is generated. The rest area outside the path corridor is regarded as an obstacle (the black area). The corridor clearance map is produced with Alg. 6.1.



(a) (b)

Figure 6.4: a) Corridor clearance map $path\_clear[]$ overlaid by normative path cost distribution $norm\_cost[]$ along P_{norm}; b) Path position can be found by repeatedly moving along direction of gradient in corridor clearance map

Fig. 6.4(a) also shows cost distribution $norm\_cost[]$ along the normative path which is generated in the same way as the global path planning. The transition of the colors in Fig 6.4(a) shows the variation of the cost along P_{norm}. For each configuration ζ_i inside the path corridor, there is a corresponding path position $p_i \in P_{norm}$. Similarly, p_i can be found by repeatedly moving from the current position along the direction of the gradient in the corridor clearance map, shown as the yellow arrows in Fig. 6.4(b). Any configuration ζ_i inside the corridor could be connected to P_{norm} with its path position p_i, and also evaluated with normative path cost $norm\_cost[p_i]$.

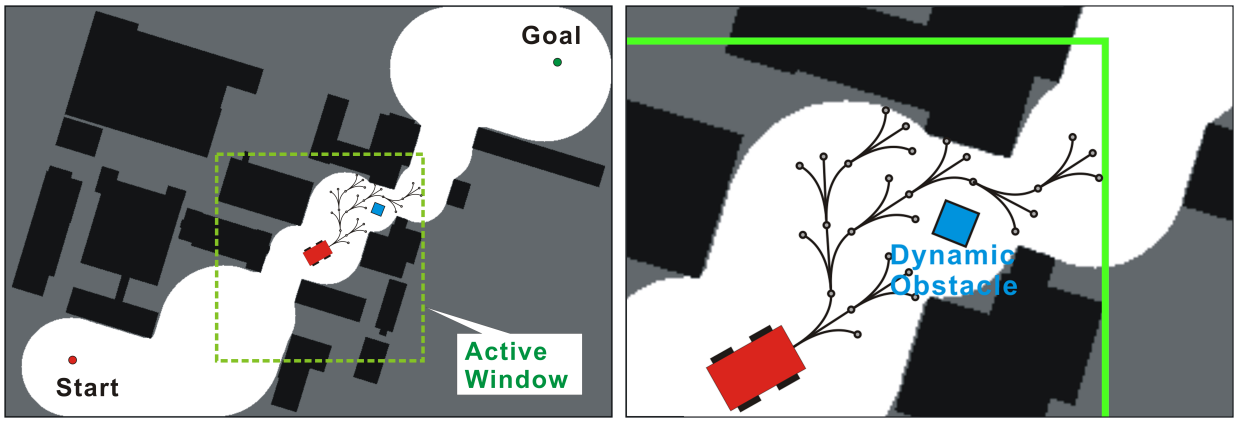
Algorithm 6.1: Corridor clearance map

clear[] : Global clearance map [Fig 6.3(a)]
path_clear[]: Corridor clearance map [Fig. 6.4(a)]
P_{norm} : Normative path
dist(,) : Function, which gets the distance between two points

```
1 Function ret GetClearanceMap(P_norm)
2     initiate all occupied in path_clear[] with 0;
3     foreach p_i ∈ P_norm do
4         foreach cell ζ_i, dist(ζ_i, p_i) < clear[p_i] do
5             clearance ← path_clear[ζ_i] − dist(ζ_i, p_i);
6             if clearance > path_clear[ζ_i] then
7                 path_clear[ζ_i] ← clearance;
8     return path_clear[];
```

6.1.4 Active Window

It is assumed that an active window [101] is attached with the robot and local path planning is limited inside the window. The active window is centered on the robot and moves along with it. The primitive trajectory-based search is applied inside the active window.



(a) (b)

Figure 6.5: Active window and local path planning

As shown in Fig. 6.5(a), the green frame is the active window attached with the robot. Primitive trajectory-based path planning is applied locally inside both the active window and the path corridor. The blue object is a dynamic obstacle that the robot encountered on the way. Local path planning is restricted inside the path corridor and the active window,

only a small area around the robot needs to be searched, largely reducing the computation cost of local path planning.

6.1.5 Normative Path-based Heuristic

For path corridor-based path planning, the nodes are still evaluated based on the A* algorithm (6.1) (Section 2.2.2).

$$f(n_i) = g(n_i) + h(n_i) \tag{6.1}$$

$g(n_i)$ is calculated in the same way as in the global path planning. However, the heuristic value $h(n_i)$ is now evaluated by normative path-based heuristic as (6.2). p_i is the path position of n_i on P_{norm} [as in Fig. 6.4(b)]. As mentioned in Section 6.1.3, for each node n_i during local path planning process, there is corresponding path position p_i on normative path P_{norm}. p_i can be found in the same way by repeatedly moving from the current position along the direction of the gradient in the corridor clearance map (this process is shown as the white arrows in Fig. 6.6).

$$NormativePathHeuristic(n_i) = norm\_cost[p_i] + \kappa_{ori}\ ori\_cost(n_i) + \kappa_{ecc}\ ecc\_cost(n_i) \tag{6.2}$$

$norm\_cost[p_i]$ in Eq. (6.2) is the normative path cost (Section 6.1.3), which represents the estimated cost from current p_i to the goal along the normative path [as in Fig 6.4(a)], where p_i is the path position of n_i. $norm\_cost[p_i]$ always guides the search towards the goal inside the path corridor.

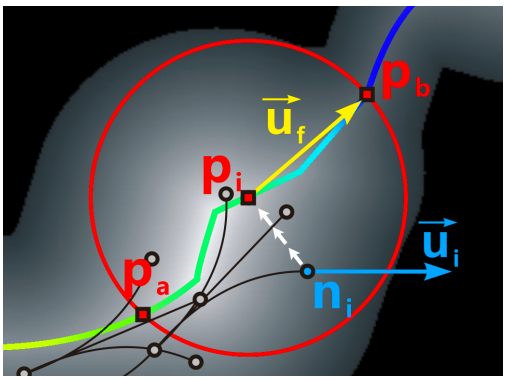


Figure 6.6: Orientation cost;

Orientation cost $ori\_cost(n_i)$ represents the orientation difference between n_i and the reference orientation of p_i. The orientation of p_i can be found by setting a red circle with the radius of $clear[p_i]$ on p_i (as shown in Fig. 6.6). p_a and p_b are the intersection points of P_{norm} with the circle. Depending on the path cost of p_a and p_b, p_b is nearer to the goal along P_{norm}. Therefore, the reference orientation of p_i is $\overrightarrow{u_f} = \overrightarrow{p_i p_b}$ (the expected orientation of the

robot on p_i). $ori\_cost(n_i)$ is calculated as Eq. (6.3), where $\langle , \rangle$ is the angle between the two vectors. The orientation cost helps the search orient along the normative path.

$$ori\_cost(n_i) = \langle \overrightarrow{u_i}, \overrightarrow{u_f} \rangle \tag{6.3}$$

$ecc\_cost(n_i)$ of Eq. (6.2) is simply the distance between n_i and its path position p_i. $ecc\_cost(n_i)$ shows how far n_i is away from the center of the corridor. Such a cost is particularly useful when the width of the corridor suddenly shrinks at a certain point along the normative path. As Fig. 6.6 shows, the corridor shrinks dramatically at p_b, which becomes a small entrance to the following section of the corridor. $ecc\_cost(n_i)$ helps rapidly guide the search to such an entrance.

κ_{ori} and κ_{ecc} are the coefficients that transform $ori\_cost(n_i)$ and $ecc\_cost(p_i, n_i)$ into a time cost.

6.2 Goal Zone

The goal zone was defined in Section 5.2.5. The goal zone is the convex area of the goal, where the heuristic function (Alg. 5.2) returns the Euclidean distance to the goal, so that the search can eventually reach the real goal configuration. Similarly, in the path corridor-based local path planning, as long as a new node is inserted into the search tree, the search will check whether n_i lands in the goal zone, i.e. whether or not GVD position χ_i of n_i meets $\chi_i \in X_{goal}$ (Section 5.2.5). If that is true, the evaluation of n_i will no longer be limited to the path corridor. In other words, if any node n_i during local path planning lands in the goal zone, n_i will be evaluated exactly in the same way as in global path planning. The only difference is that n_i still has to be limited in the active window that includes the dynamic obstacles. This endues the local path planner with more flexibility, as the search approaches the neighborhood of the goal.

6.3 Blocked Corridor

In most cases, the robot is capable of avoiding the obstacles inside the path corridor. However, it happens sometimes [as in Fig. 6.7(a)], the yellow objects are the obstacles blocked the whole corridor. In such cases, the robot can no longer pass through the path corridor and the global path needs to be regenerated.

6.3.1 Blocked Corridor Detection

Such a situation can be detected with Alg. 6.2. While the robot is moving in the corridor, it keeps updating its path position p_{robot} based on its current position. p_{min} is the furthest path position that the robot currently has reached along normative path P_{norm}. If $norm\_cost[p_{robot}] < norm\_cost[p_{min}]$ (line 7), it means the robot is making progress towards

Algorithm 6.2: Blocked corridor detection

p_{min} : Furthest path position that the robot currently has reached along normative path P_{norm}
p_{start} : First path position of normative path P_{norm}
p_{robot} : Current path position of robot
Φ_{clock}: Clock which will be reset to 0 every time p_{min} is changed
Ω_t : Time threshold used to decide whether or not the corridor is blocked

```
1  Function ret Blocked_Corridor_Detection(Ω_t)
2      p_min ← p_start;
3      reset Φ_clock;
4      loop
5          update p_robot;
6          // Whether robot is making progress along nomative path.
7          if norm_cost[p_robot] < norm_cost[p_min] then
8              p_min ← p_robot;
9              reset Φ_clock;
10         // Whether robot has not made progress for too long.
11         else if Φ_clock > Ω_t then
12             // Corridor is blocked.
13             return blocked;
```

the goal, the p_{min} is also updated with p_{robot}. Every time p_{min} is set with a new path position, clock Φ_{clock} will be reset to 0 accordingly. Therefore, if $\Phi_{clock} > \Omega_t$ (line 11), it means the robot has not made any progress for too long. Therefore, the path corridor is then regarded as blocked. Here, Ω_t is a time threshold that used to decide whether the corridor is blocked, and whether a new global path should be regenerated.

6.3.2 Global Path Reproduction

As long as the corridor is detected as blocked, global path planning will be applied on the global map combined with the detected dynamic obstacles in the current active window. As shown in Fig. 6.7(b), the Voronoi-based heuristic would still attract the search towards the block area at first, resulting in the accumulation of the exploring tree around $\chi_{blocked}$ on the GVD (somewhere in front of the dynamic obstacles). As mentioned in Section 5.3.2, this reduces the clearance value of the GVD section around $\chi_{blocked}$ and consequently triggers the regeneration of the cost distribution over the GVD. Since the reduction of the clearance value increases the cost around $\chi_{blocked}$, eventually the Voronoi-based heuristic would guide the search away from the block area [as in Fig. 6.7(b)] and another path corridor [as in Fig. 6.7(c)] is produced.

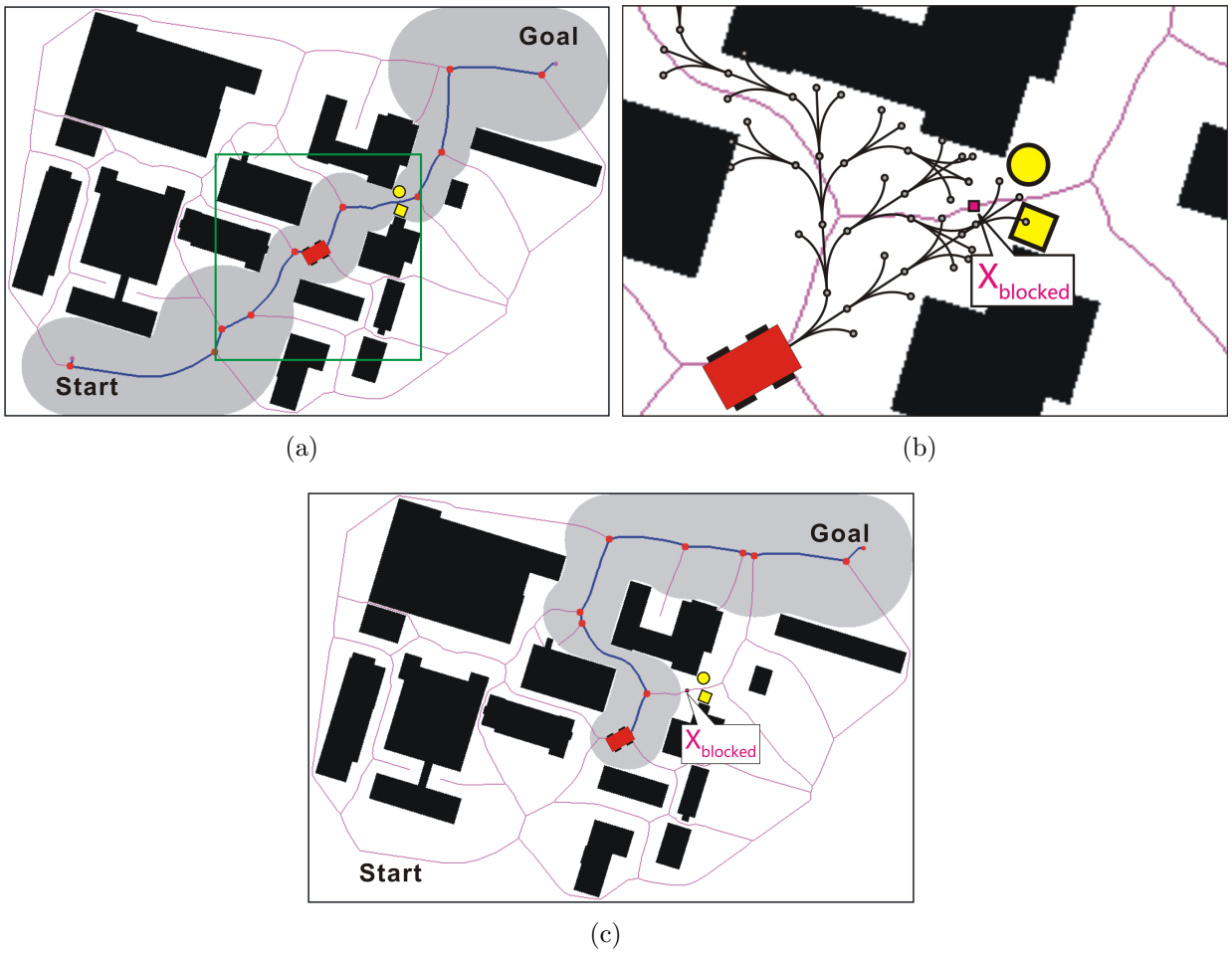


Figure 6.7: a) Blocked corridor; b) Inserted break point; c) Regenerated path based on inserted break point

6.4 Summary

This chapter proposed a path corridor-based local path planning algorithm. Rather than a fixed global path, the planner extracts normative path P_{norm} and a path corridor from the global path. The path corridor defines the maximum space along the normative path, giving the robot enough flexibility to deal with the dynamic obstacles. The computation cost is largely reduced by restricting local path planning inside the path corridor. In such a manner, the local path can be updated in real time, efficiently avoiding the dynamic obstacles. For situations where the whole path is blocked by obstacles, the global path will be regenerated and avoid the blocked spot.

Part IV

Simulation and Evaluation

Chapter 7

Evaluation of Experiments

In this chapter several experiments are done to demonstrate the usefulness of the proposed approaches. The simulation and the algorithms are programmed with C++ and tested on a computer with an Intel Core Duo(TM)2 CPU P8700 2.53 GHz and 4GB RAM under 64 bit Windows operating system.

7.1 Evaluation of Voronoi-based Parallel Thinning Algorithm

As introduced in Section 2.1.2, GVD is the collection of medial lines and precisely defines the connectivity of the free space. Rather than estimating the heuristic cost from the current location to the goal with the Euclidean distance, the cost to the goal is measured along the GVD-based roadmap.

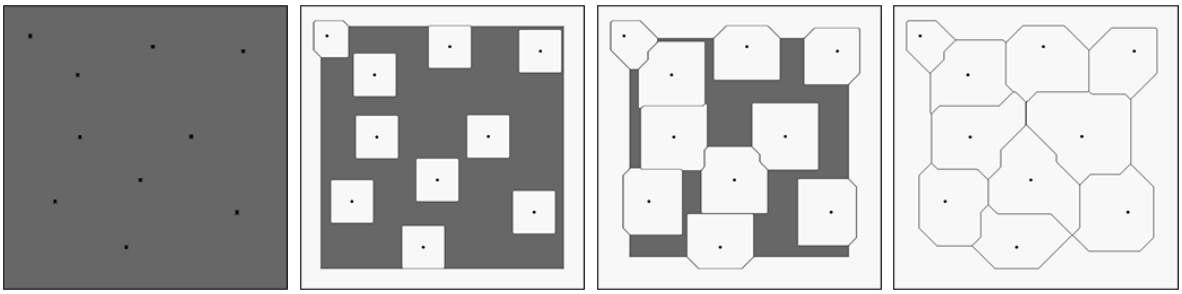
However, as addressed in Section 4.4, the classic parallel thinning algorithm (CPT) [53] has two drawbacks and cannot produce an accurate GVD. The objective of this experiment is to show the effectiveness of the proposed Voronoi parallel thinning algorithm (VPT) in Chapter 4. The experiment shows that the VPT is fully capable of producing an accurate GVD, by overcoming the drawbacks of the classic thinning algorithm.

7.1.1 Square Effect

[1]The CPT and VPT algorithms are tested on the same grid map [as Fig. 7.1(a)], which is a simple blank square area with a few black dot objects.

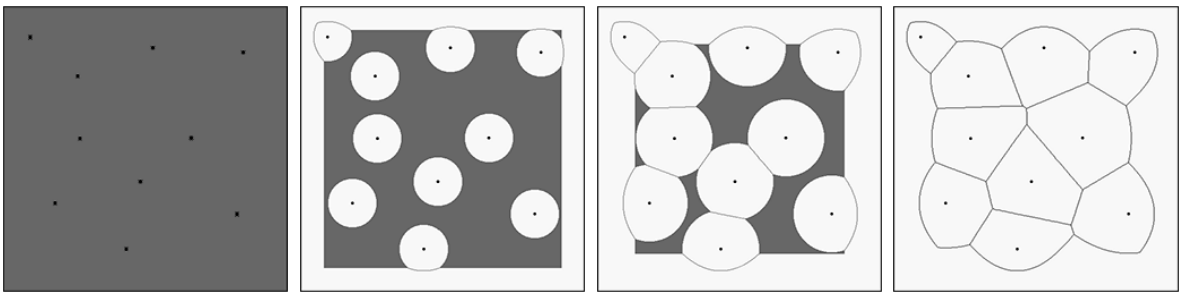
Fig. 7.1(a)∼7.1(d) show the process of the CPT algorithm. The gray space represents the unprocessed space. It is observable that the gray space shrinks until a very thin line is left. However, the thinning process of the CPT shows a very clear square effect caused by the layout of the grid map. The gray area shrinks in diagonal directions faster than in

[1]The video of this experiment is available at `https://youtu.be/3KPEaPiIhjw`



(a) (b) (c) (d)

Figure 7.1: Classic parallel thinning



(a) (b) (c) (d)

Figure 7.2: Voronoi-based parallel thinning

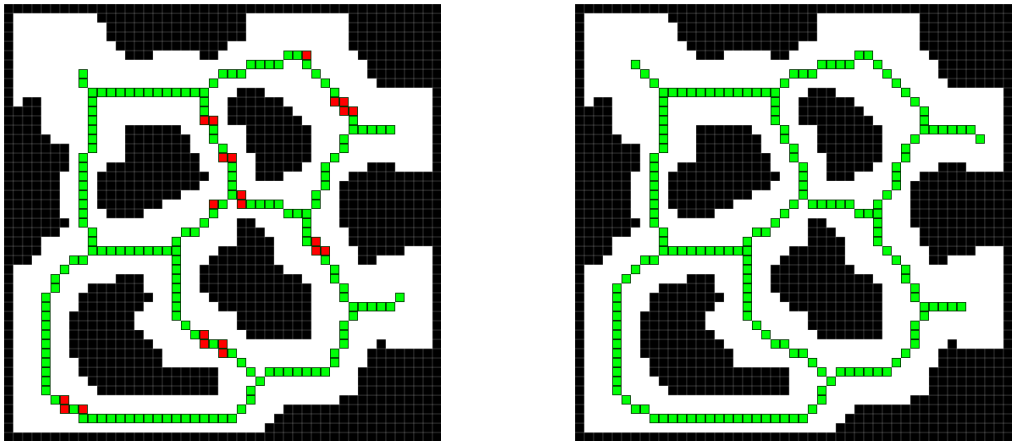
vertical and horizontal directions, the produced GVD is exactly on the medial line between the obstacles and, therefore, the result of the CPT shown in Fig. 7.1(d) does not represent the true GVD.

Fig. 7.2(a)∼7.2(d) show the process of the proposed VPT algorithm. Fig. 7.2(b) and Fig. 7.2(b) show that the gray space equally shrinks in all directions and shows a circular effect. Therefore, the VPT eventually produces an accurate and smoothed GVD in Fig. 7.2(d).

7.1.2 Single Cell-connected GVD

As mentioned in Chapter 4, the CPT algorithm produces a GVD with some non-single cell-connected spots that pose ambiguities when calculating the distance along the GVD. The ambiguities can result in inaccuracy of the produced Voronoi-based heuristic. Fig. 7.3(a) is the thinning result without the integration of the supplementary patterns proposed in Section 4.5.2. The green cells show the locations of the extracted GVD and the non-single cell-connected spots are marked as red cells.

Fig. 7.3(b) is the result after the integration of supplementary patterns (Section 4.5.2),



(a) (b)

Figure 7.3: a) Thinning result without supplementary patterns; b) Thinning result with supplementary patterns

which clearly shows that all non-single cell-connected spots on the GVD disappeared after the integration. This avoids the ambiguity of measuring the cost along the GVD.

7.2 Evaluation of Voronoi-based Heuristic

In order to demonstrate the usefulness of the Voronoi-based heuristic, it is compared with Euclidean heuristic and informative heuristic in the following sections. As mentioned in Section 2.3.1, the Euclidean heuristic it one of the most classic heuristics. The Euclidean heuristic is simply the distance between the goal and the current position regardless of the obstacles, it makes the search easily trapped by the local minimum. The informative heuristic was introduced in Section 2.3.3, which integrates with the information of obstacles constraints, however, as it will be shown in the following experiment in Section 7.2.2, the informative heuristic is still unable to prevent the search from being trapped by the local minimum that is caused by both obstacle constraints and nonholonomic constraints. Additionally, since the application of informative heuristic requires a 2D Djistra's grid map search to be done first, which result in an extra overhead that increases the computation cost.

7.2.1 Comparison with Euclidean Heuristic

The Voronoi heuristic-based search (VHS) and Euclidean heuristic-based search(EHS) are applied to three different grid maps: A. without local minima (Fig. 7.4); B. with shallow local minima (Fig. 7.5); C. with deep local minima (Fig. 7.6); The start and the goal configuration are shown as red and green arrows, respectively. The start and the goal are

set far apart and the search has to overcome all obstacles to reach the goal. This experiment shows how the local minimum can increase the computation cost of the search.

For each scenario, primitive trajectory set-based searches (Chapter 3) are applied with the EHS and the VHS in a discrete 3D configuration space, $S^d = \{x_i, y_i, \theta_i\}$. The directional dimension θ_i is discretized to $K = 32$ entries as mentioned in Section 3.1.

Map type	No local minimum	Shallow local minimum	Deep local minimum
Grid map size	500×300	500×300	500×300
GVD extraction	138.70 ms	178.75 ms	191.16 ms

Table 7.1: Size of grid map and computation time of VPT-based GVD extraction

Tab. 7.1 shows the size of the grid maps and the computation time of their GVDs with VPT. As long as the GVD is extracted, the proposed Voronoi-based heuristic can be used for a multi-query search. Therefore in following sections, the computation time of VPT is not included in the computation time of the VHS. However, Tab. 7.1 shows that the computation of the GVD only requires just a few hundreds of milliseconds. This shows the GVD extraction is very fast, hence, the Voronoi-based heuristic can also be used for a single-query search where the computation time of VHS comprises the time cost of VPT.

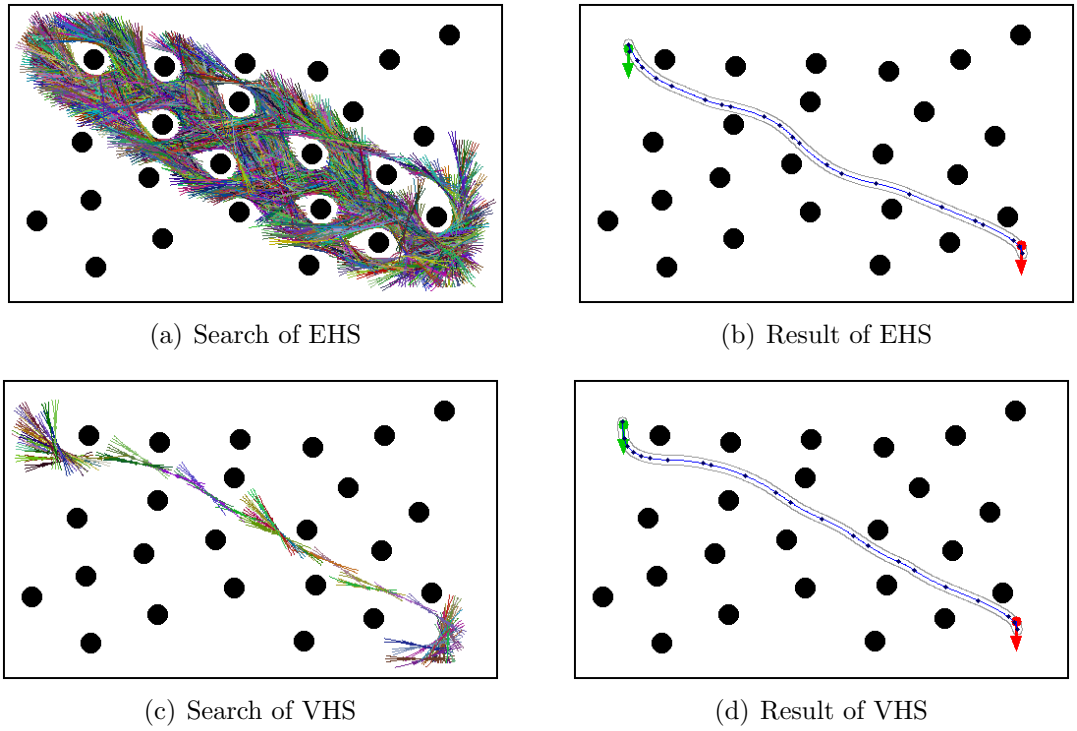
No Local Minimum

Fig. 7.4 is an environment that only consists of convex obstacles, i.e. no local minimum. Tab. 7.2 shows different aspects of the VHS and the EHS in this scenario. "Computation time"represents the time cost to produce a feasible path (the computation of VPT is not included). "Memory Cost"is the measurement of the dynamic objects used to create the exploring tree during the search. "Path cost"represents the evaluation of the resulting path as introduced in Section 3.3. The last column of Tab. 7.4 is the percentage of the value of the "VHS"over the "EHS", which shows the comparison between them.

	VHS	EHS	VHS/EHS
Computation time	3.75 ms	584.21 ms	0.64%
Memory cost	975	28103	3.47%
Path cost	541.56	538.22	100.62%

Table 7.2: No local minimum

From Tab. 7.2, it is observable that the VHS performs more effective than the EHS except a little drop back of the path cost. In Fig. 7.4(a), the EHS almost fully searched the area between the start and the goal. In comparison, the VHS has only searched a few steps [as in Fig. 7.4(c)], and the produced paths are nevertheless equally optimal [Fig. 7.4(b) and 7.4(d)]. The same result can also be seen from Tab. 7.2, the path cost of the VHS is only 0.62 percent more than the EHS, but the computation time and the memory cost dramatically decreased by 99.36 percent and 96.53 percent respectively.



(a) Search of EHS (b) Result of EHS

(c) Search of VHS (d) Result of VHS

Figure 7.4: No local minimum

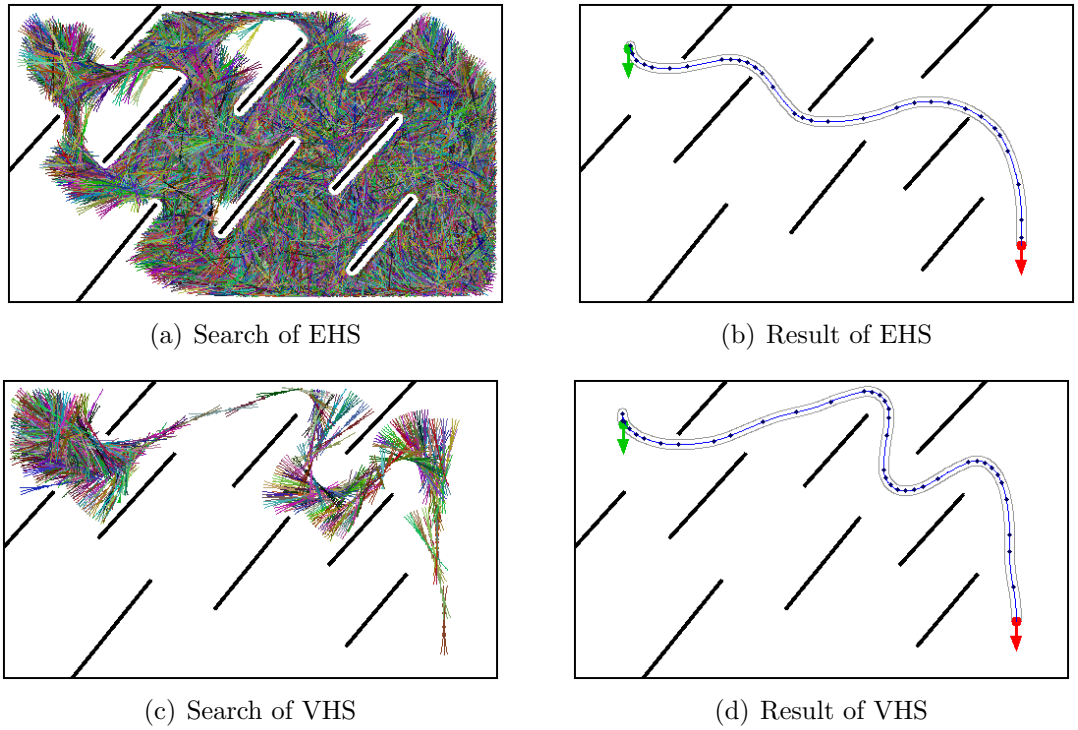
Shallow Local Minima

There are several stick-shaped obstacles in Fig. 7.5 that form many shallow minima. In this scenario, the strength of the Voronoi-based heuristic becomes more dominant. The EHS has to search almost the entire space [as in Fig. 7.5(a)], making the EHS very time consuming and requiring a very high memory cost [as in Tab. 7.3].

On the contrary, the VHS nicely navigates through the obstacles without being trapped at all. It demonstrates more intelligent behavior (as in Tab. 7.3). The computation time as well as the memory cost of the VHS are only a very small fraction of the comparatives for the EHS. Comparing the resulting paths of the two searches, the path cost of the VHS is 32.14 percent more than that of the EHS. However, the result in Fig. 7.5(d) shows that the quality of the produced path of the VHS is still in an acceptable range.

	VHS	EHS	VHS/EHS
Computation time	29.08 ms	98229.97 ms	0.03%
Memory cost	4867	138550	3.51%
Path cost	768.79	581.78	132.14%

Table 7.3: Shallow local minima



(a) Search of EHS (b) Result of EHS

(c) Search of VHS (d) Result of VHS

Figure 7.5: Shallow local minima

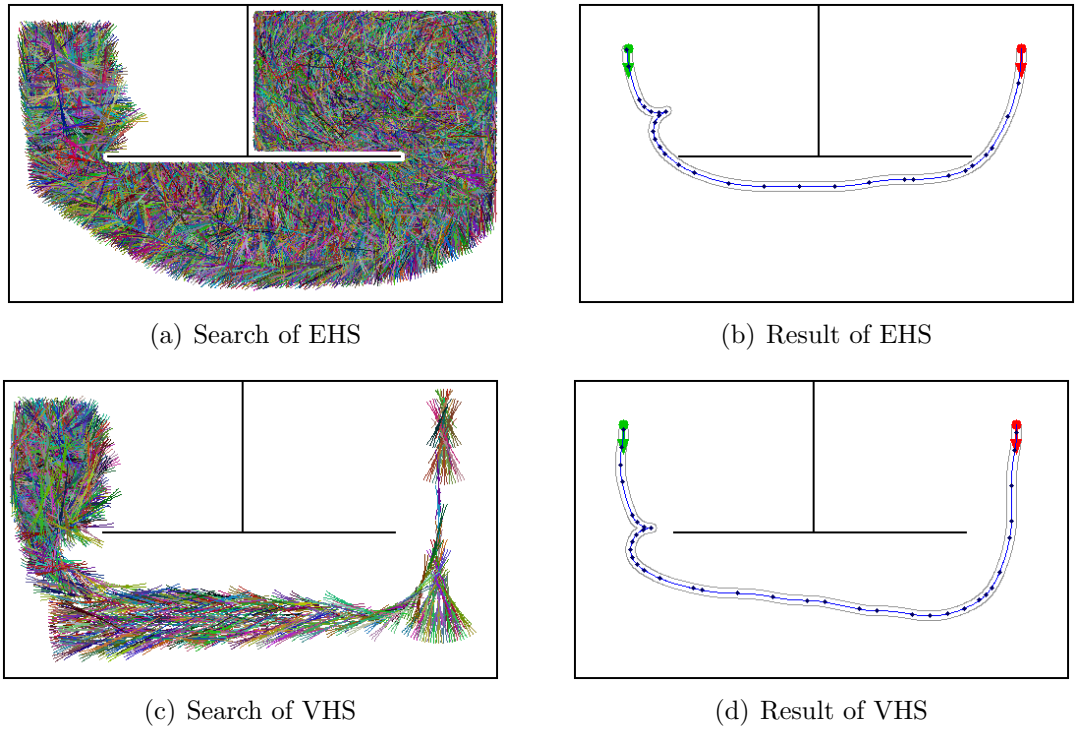
Deep Local Minimum

The large convex area between the goal and the start (as in Fig. 7.6) forms a deep local minimum. The deep local minimum can cause serious problems for the EHS since the search will be trapped in the deep local minimum almost eternally and may easily make the computation run out of memory. The EHS costs almost 16 minutes (as in Tab. 7.4) whereas the VHS only costs 110.51 milliseconds or only 0.12 percent of the EHS.

	VHS	EHS	VHS/EHS
Computation time	110.51 ms	95339.28 ms	0.12%
Memory cost	11828	163146	7.25%
Path cost	736.74	624.34	118.00%

Table 7.4: Deep local minimum

From Fig. 7.6(a), it is quite clear how much the EHS can be trapped since the entire convex area has been fully searched. The computation cost of the EHS directly depends on the size of the convex area. It is inferable that the larger the convex area, the longer the EHS will be trapped. The VHS nicely solves this problem. As shown in Fig. 7.6(c), the VHS immediately found its way under the guidance of the Voronoi-based heuristic and directly



(a) Search of EHS

(b) Result of EHS

(c) Search of VHS

(d) Result of VHS

Figure 7.6: Deep local minimum

left the convex area right after the search began, greatly reducing the computation time and the memory cost with only a minor sacrifice of the path cost.

7.2.2 Comparison with Informative Heuristic

Two experiments are constructed in this section to compare the VHS with the informative heuristic-based search (IHS) [93] [in Section 2.3.3] under grid maps of two different environments. The first experiment is made under the environment shown in Fig. 7.7, which is a simulated shopping center defined in Appendix A.2, where the robot is expected to move freely and provides services. The size of the grid map is 750×600, and it does not consist of any nonholonmoic local minimum (NLM) (Section 5.3) as shown in Fig. 7.7. The NLM is a new type of local minimum formed by the combination of the obstacles and the nonholonomic constraints defined in Section 5.3.

The second experiment shows the environment, where two open areas are connected by several narrow corridors with sharp corners in the way as shown in Fig. 7.8 and Fig. 7.9. The sharp corner violates the nonholonomic constraints of the car-like robot and, thus, forms NLM that make the corresponding corridor no longer traversable. Such scenario appears often in a supper market or a warehouse where the robot has to navigate through the narrow corridors that sometimes may violate its nonholonomic constraints.

The Voronoi-based heuristic shows better results than IHS in both environments. However, as the test in Section 7.2.2 shows, the strength of Voronoi-based heuristic over informative heuristic is more dramatic on the grid map with NLM.

No NLM

[2]The VHS, EHS and IHS [93] are applied to the grid map of a clustered environment (Fig. 7.7), The start and the goal configurations are shown as red and green arrows respectively. After the GVD is extracted, the proposed Voronoi-based heuristic can be used for a multi-query search as long as the grid map is not changed. Additionally, the GVD extraction with VPT is very fast, the computation of VPT on the grid map in Fig. 7.7 only cost 357.90 ms.

	EHS	IHS	VHS
Computation time	11405.86 ms	717.02 ms	76.84 ms
Memory cost	79740	10680	9486
Path cost	133.19	162.93	156.80

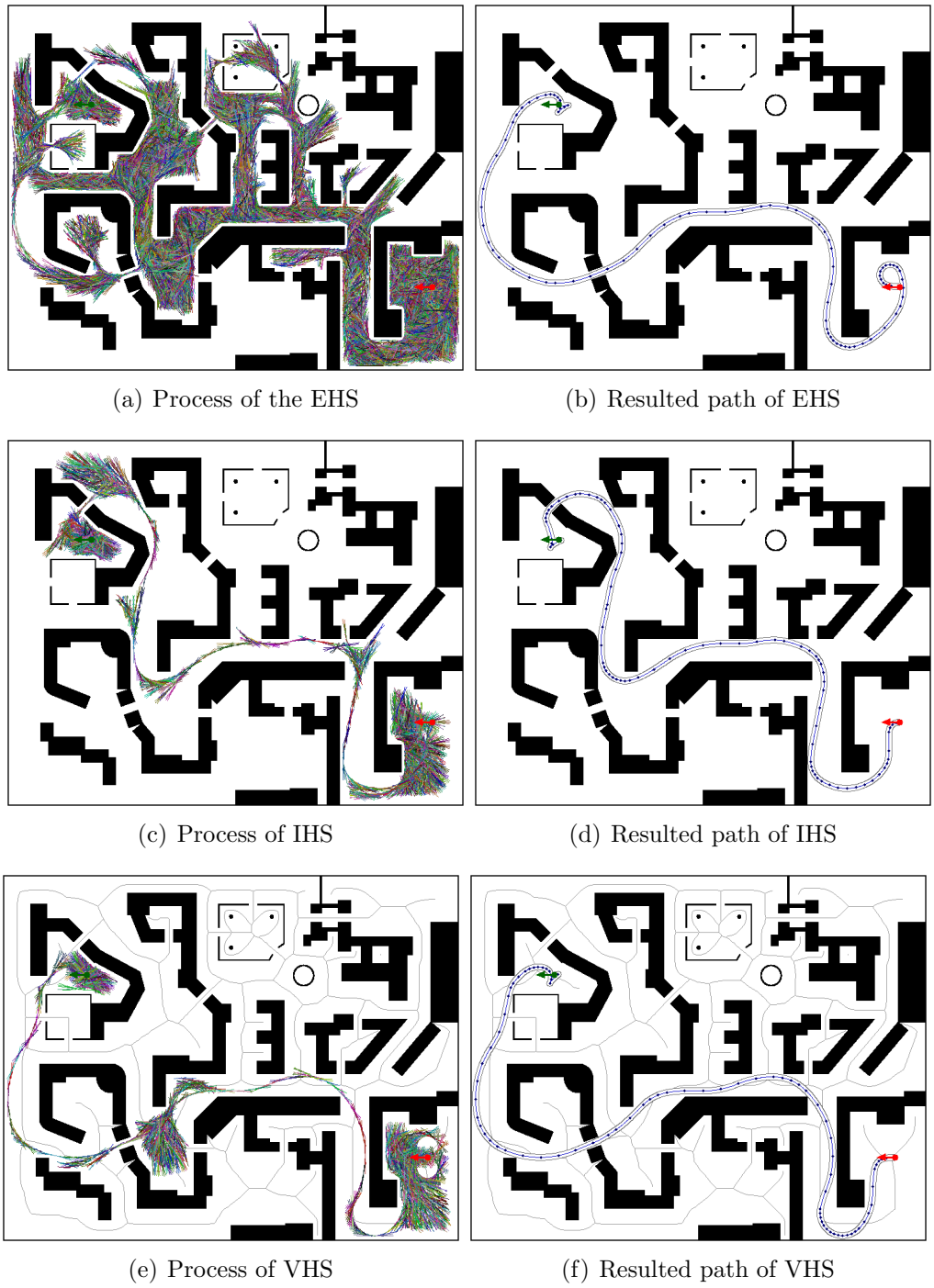
Table 7.5: Clustered environment

The local minimum appears quite often in a clustered environment, which can make the EHS dramatically trapped. This is proven by the result of EHS shown in Fig. 7.7(a) and Fig. 7.7(b). The entire space is almost fully explored [Fig. 7.7(a)]. Since the Euclidean-based heuristic directly attracts the search to move toward the goal regardless of the obstacles, the EHS has to explore fully the local minimum between the start and the goal. This result in both large amount of computation time and memory cost as shown in Tab. 7.2.2.

In contrast, the IHS and VHS [as in Fig. 7.7(c) and Fig. 7.7(e)] directly navigate through the space to the goal. This is because both the informative heuristic and Voronoi-based heuristic are generated by following the obstacle constraints. The gray line in Fig. 7.7(e) and Fig. 7.7(f) are the generated GVD through the application of VPT on the grid map. As shown in Tab. 7.2.2, the computation time and memory cost of the IHS and the VHS are dramatically reduced, although their path costs are slightly higher than EHS, which renders the resulted paths a little less optimal.

It is also observable from Tab. 7.2.2 that the memory costs of the IHS and the VHS are almost the same, which means the IHS and the VHS explore approximately the same number of nodes during the search, but the computation time of IHS is 9 times higher than that of the VHS. As mentioned earlier, the computation of IHS comprises two parts. The first part is the 2D grid map search that attaches a heuristic value to each cell, while the second part is the motion primitive-based search [Fig. 7.7(c)] that is guided by the heuristic value and produces the actual path [Fig. 7.7(d)]. The motion primitive-based search of IHS costs only 76.77 ms, whereas the 2D grid map search costs 640.25 ms. This makes the total computation time of IHS up to 717.02 ms.

[2]The video of this experiment is available at `https://youtu.be/D7tg2qSXYTo`



(a) Process of the EHS (b) Resulted path of EHS

(c) Process of IHS (d) Resulted path of IHS

(e) Process of VHS (f) Resulted path of VHS

Figure 7.7: Processes and resulted paths of EHS, IHS and VHS under clustered environment

As mentioned in Section 5.2.2, the VHS also requires a cost distribution along GVD. However, its computation time is merely about 0.99 ms, as the GVD is a small subset of the grid map. For multiple queries based on the same grid map, the GVD only needs to be produced once with VPT. Therefore, the computation time of the VHS for multiple quires does not include the computation time of VPT. The computation of VPT on the 750×600 grid map in Fig. 7.7 is 357.90 ms. Thus, for the single-query search of the VHS including the computation of VPT, its total computation time is merely 434.74 (357.90 + 76.84) ms, which is even far less than the computation time of the 2D grid map search of the IHS.

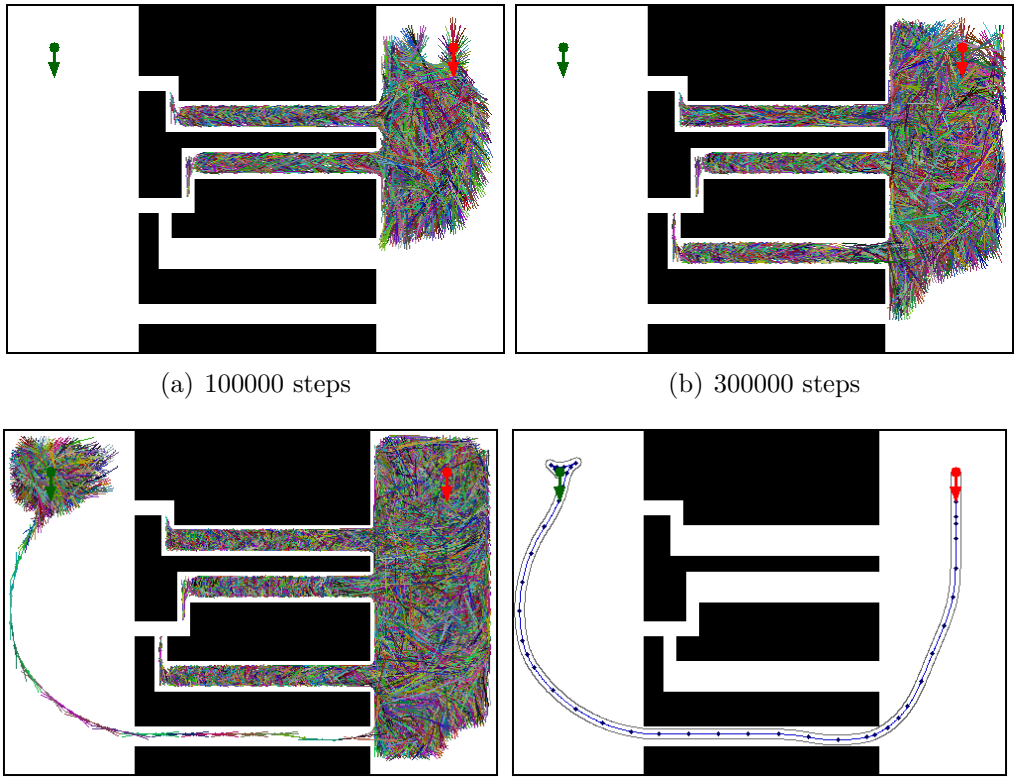
With NLM

[3]In the above experiment, both the Voronoi-based heuristic and informative heuristic guide the search through obstacles effectively. However, it will be shown in this section, the informative heuristic only depends on obstacle constraints, which can still make the IHS trapped by the nonholonomic local minimum (NLM). As introduced in Section 5.3, the NLM is formed by the combination of the obstacles and the nonholonomic constraints. The objective of this experiment is to demonstrate how much the NLM can increase the computation cost of the IHS and how this problem can be solved by the Voronoi-based heuristic proposed in Section 5.3.2.

Fig. 7.8 shows the result of the IHS. Several corridors lead to the goal. Except for the bottom-most corridor, all corridors include a sharp turning in the way. The sharp turnings violate the nonholonomic constraints of the mobile robot making the corridors untraversable. However, the informative heuristic cost along the upper corridors is lower than that along the bottommost corridor since the path through the upper corridors is shorter. Therefore, the informative heuristic keeps attracting the search toward the upper corridors [as in Fig. 7.8(a) and 7.8(b)]. As the search continues, the exploring tree starts to accumulate and sequentially search the corridors from the top to the bottom. By the time the search eventually reaches the bottom corridor that meets the nonholonomic constraints [as in Fig. 7.8(c)], the open area on the right side has been explored fully, which clearly requires a large amount of computation cost. Fig. 7.8(d) is the resulting path of the IHS. In a clustered environment, the NLM appears very often and poses another type of local minimum that can also seriously trap the search.

Fig. 7.9 illustrates the test of VHS applying the NLM avoidance during the search (Section 5.3). The detection of the NLM is done with Alg. 6 as proposed in Section 5.3.2. At first, the Voronoi-based heuristic also tries to attract the search toward the top-most corridor. However, as long as the search reaches the sharp corner which violates the nonholonomic constraints and forms an NLM for the search, it can no longer explore forward and, therefore, immediately starts to accumulate around the NLM [as shown in Fig. 7.9(a)]. As mentioned in Section 5.3.3, any new creation of the node in the exploring tree will slightly reduce the clearance value along its neighboring GVD positions (Alg. 5.3, line 6). Observably,

[3]The video of this experiment is available at `https://youtu.be/OnkOSWv4RDQ`



(a) 100000 steps (b) 300000 steps

(c) 782056 steps (d) Resulted path

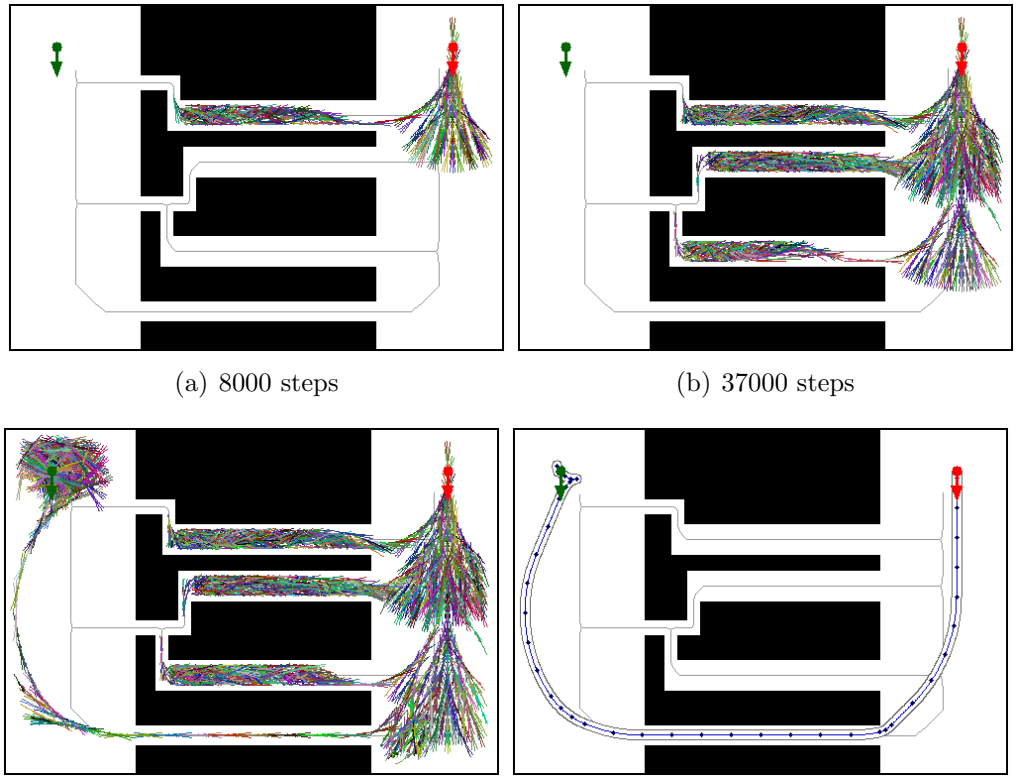
Figure 7.8: Process and resulted path of IHS under Environment with NLM

	VHS	IHS	VHS/IHS
Computation time	97.86 ms	14239.67 ms	0.69%
Memory cost	10987	74835	14.68%
Path cost	86.27	86.31	99.95%

Table 7.6: Nonholonomic local minimum test

the accumulation of the exploring tree creates nodes intensively around the sharp conner, thereby dramatically reducing the clearance value along the neighboring GVD sections (Alg. 5.3, line 7). It eventually causes the clearance value of one of the GVD positions to drop to 0 and is thus detected as an NLM (Alg. 5.3, line 7).

As mentioned earlier, the detection of the NLM triggers the regeneration of cost distribution over the GVD (Alg. 2.5, line 27). Since clearance values along the neighboring GVD section around NLM are dramatically reduced, and very low clearance values will result in



(a) 8000 steps (b) 37000 steps

(c) 56454 steps (d) Resulted path

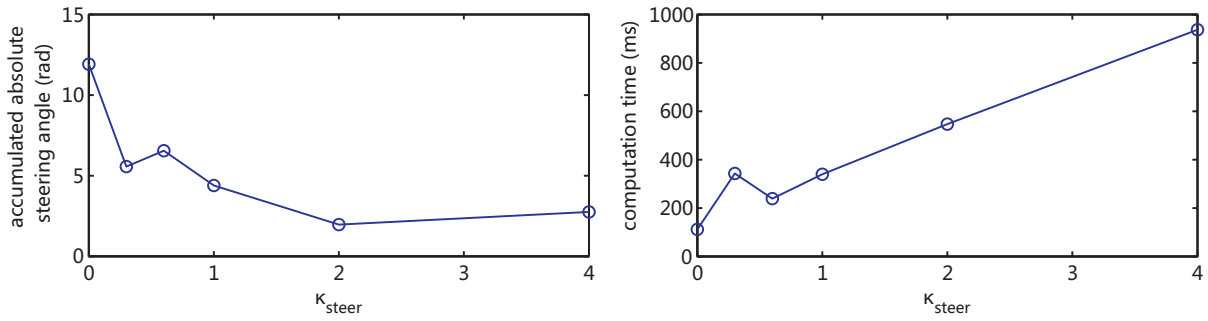
Figure 7.9: Process and resulted path of VHS under Environment with NLM

very high costs during cost distribution (Section 5.2.3), this makes the cost along the first corridor higher than the cost along the second topmost corridor in Fig. 7.8(a). Consequently, after the regeneration of cost distribution, the Voronoi-based heuristic guides the search to explore the second topmost corridor [as Fig. 7.9(b)]. A shown in Fig. 7.9(c), the second and the third topmost corridors each also include an NLM. Therefore, the same process is repeated until all NLMs are detected, and eventually the search finds its way to the goal through the bottom-most corridor. Fig. 7.9(d) shows the resulted path of the VHS.

Tab. 7.2.2 shows the computation results of the IHS and VHS. It can be seen that the IHS requires a very long computation time and a very high memory cost. The produced paths of the IHS and VHS have nearly the same path cost. However, comparing with the computation cost of IHS, the computation time and the memory cost of the VHS are reduced by 99.31 percent and 86.35 percent respectively, since the VHS is able to adaptively modify the heuristic cost during the search when a NLM is detected.

7.3 Evaluation of Steering Rate Cost

The objective of this experiment (Fig. 7.11) is to evaluate how the steering rate cost can affect the result of path planning. The cost (proposed in Section 3.3.1) is used to minimize the variation of steering and increase the smoothness of the produced path. Several experiments are done by increasingly setting steering rate coefficient $\kappa_{steer} = 0,\ 0.3,\ 0.6,\ 1,\ 2,\ 4$.



(a) (b)

Figure 7.10: a) Accumulated absolute steering angle; b) Computation time

Fig. 7.11 shows the resulting paths of setting $\kappa_{steer} = 0,\ 0.6,\ 1,\ 4$. The gray lines between the obstacles are the extracted GVD. As κ_{steer} increases, the produced paths also become increasingly smoother. When $\kappa_{steer} = 0$, the produced path in Fig. 7.12(a) is very bumpy, whereas Fig. 7.12(f) shows a very smoothed path by setting $\kappa_{steer} = 4$.

The steering angle applied along the resulting paths is shown in Fig. 7.12 where the vertical axe represents the steering angle and the horizontal axe is the displacement along the path. It can be observed that without the integration of the steering rate cost [$\kappa_{steer} = 0$ as in Fig. 7.12(a)], the produced path requires the robot to frequently perform large-scale steering. As mentioned in Section 1.2, this not only costs a great deal of energy but also damages the steering system. Moreover, it can also be very hard for the controller to maintain the robot safely on the path. As κ_{steer} increases, the effect of steering rate becomes more evident [as in Fig. 7.12(b) $\sim$ 7.12(f)]. In Fig. 7.12(f), especially, where $\kappa_{steer} = 4$, the steering becomes very smooth. Only small amount of steering is required on the way except at the beginning and the end of the path where the robot has to inevitably apply large-scale steering to quickly adjust the direction.

The accumulated absolute steering angle is the measurement of the total steering angle along the whole path. Fig. 7.10(a) shows the accumulated absolute steering angles along the path with different settings of κ_{steer}. As κ_{steer} increases, the absolute steering angle dramatically decreases to a very low level. This means the robot can reach the goal only with a small amount of steering. However, the increase of κ_{steer} also increases the computation time. The computation time approximately grows linearly with the increase of κ_{steer} [as in

(a) $\kappa_{steer} = 0$ (b) $\kappa_{steer} = 0.6$

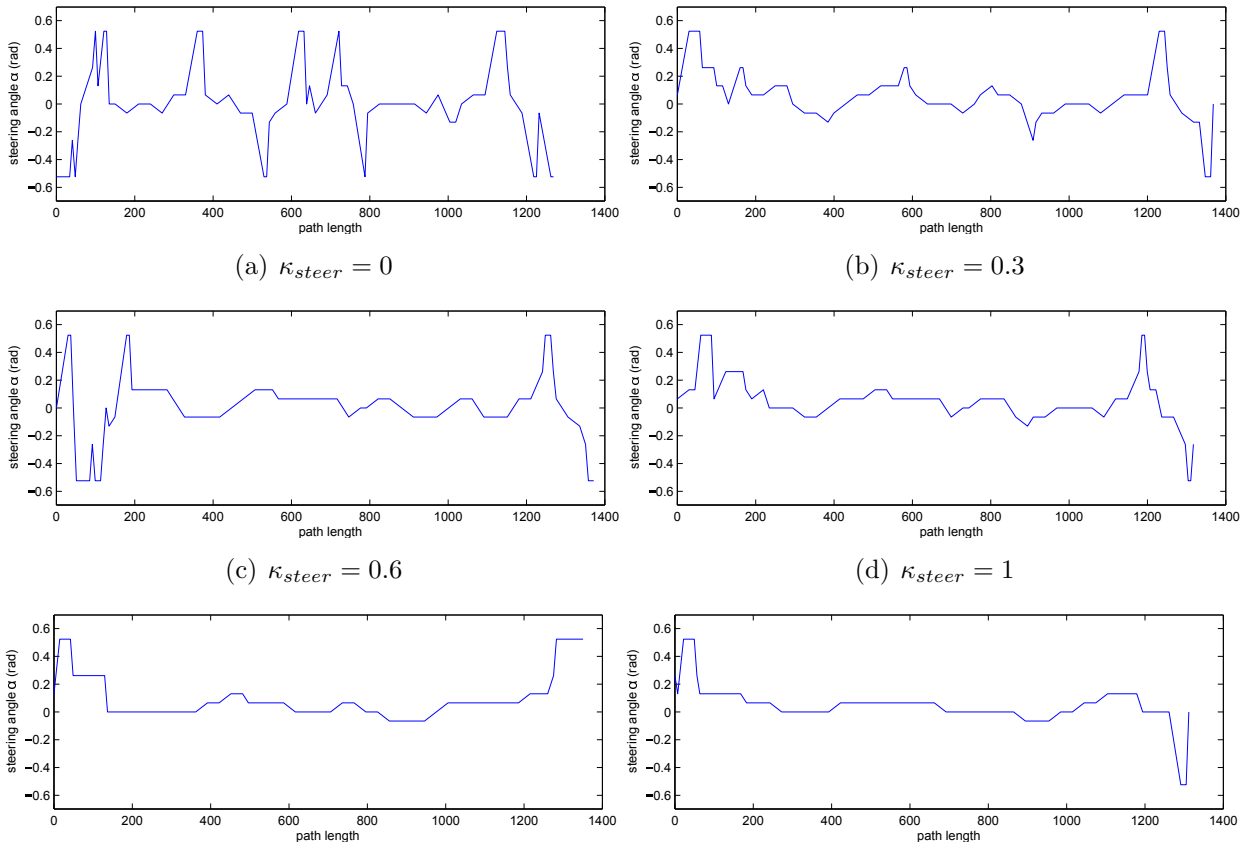
(c) $\kappa_{steer} = 1$ (d) $\kappa_{steer} = 4$

Figure 7.11: Produced global paths with different settings of κ_{steer}

Fig. 7.10(b)]. Setting $\kappa_{steer} > 2$ still increases the computation cost [as in Fig. 7.10(a)] despite not decreasing the absolute steering angle further.

7.4 Evaluation of Path Corridor-based Local Planner

The proposed path corridor-based local path planner are tested under two different environments. The first experiment compares the result of the proposed local planner with that of path following and obstacle avoidance (SPFOA) [22, 23, 113–116] (Section 2.5). The second experiment tests the path corridor-based local path planner under a more sophisticated environment with dynamic obstacles, which shows the overall performance of the proposed planner.



(a) $\kappa_{steer} = 0$ (b) $\kappa_{steer} = 0.3$

(c) $\kappa_{steer} = 0.6$ (d) $\kappa_{steer} = 1$

(e) $\kappa_{steer} = 2$ (f) $\kappa_{steer} = 4$

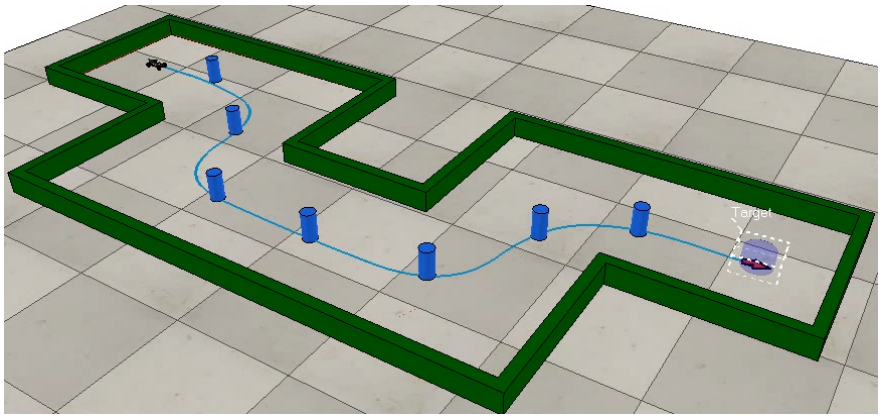
Figure 7.12: Variation of steering angle along path

7.4.1 Comparison with SPFOA

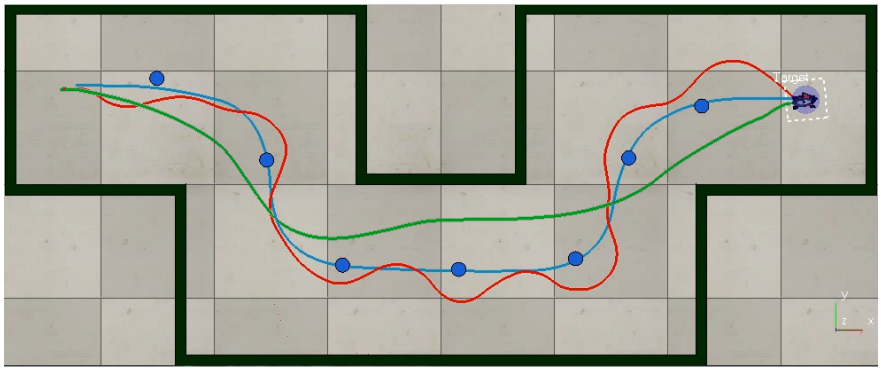
As shown in Fig. 7.14, the light blue line on the ground represents a predefined global path and the cylinders represent dynamic obstacles on it. The SPFOA-based local planner [22,23,113–116] has to follow the global path (the light blue line) and avoid dynamic obstacles (the cylinders) in the way.

As shown in Fig. 7.13(b), the red line shows the result of the SPFOA-based local planner. It is observable that since the robot tries to avoid obstacles while following the global path, it has to frequently leave and return to the global path, which results in a very snaky path as mentioned Section 1.2. The green line is the result of applying the proposed path corridor-based local planner. As the proposed planner does not required to follow any specific global path, the result of the path corridor-based planner becomes very smooth and natural.

The results of SPFOA and path corridor-based planner can be also compared in Tab. 7.7. "Traveled Distance"is the length of the resulted path, and "Total Steering"is the total absolute steering that the robot applies along the path. Tab. 7.7 shows that the proposed



(a)



(b)

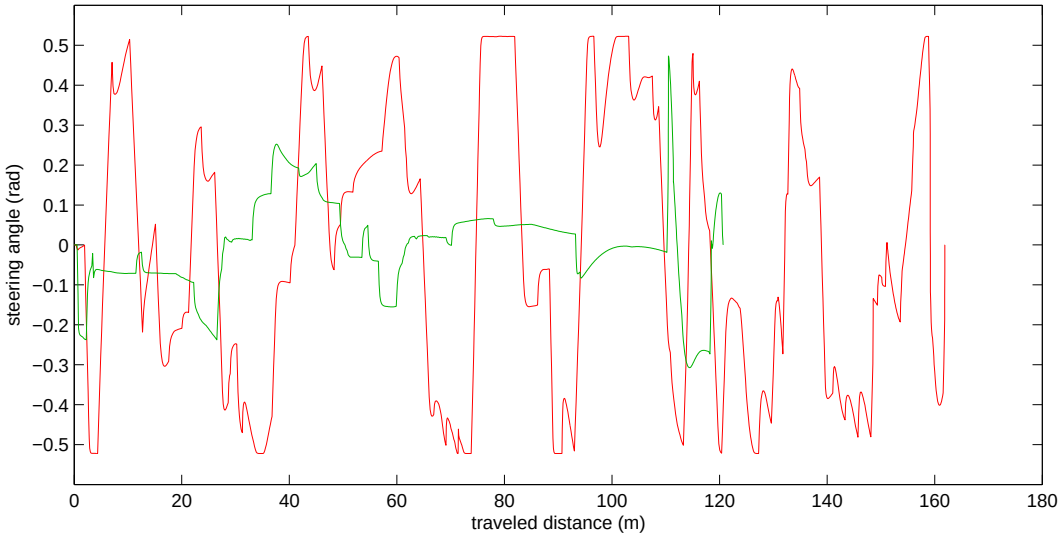
Figure 7.13: Resulted paths of SPFOA and path corridor-based local planner

	Traveled Distance (m)	Total Steering (rad)
SPFOA	163.08	22.65
Path corridor-based	120.68	4.45

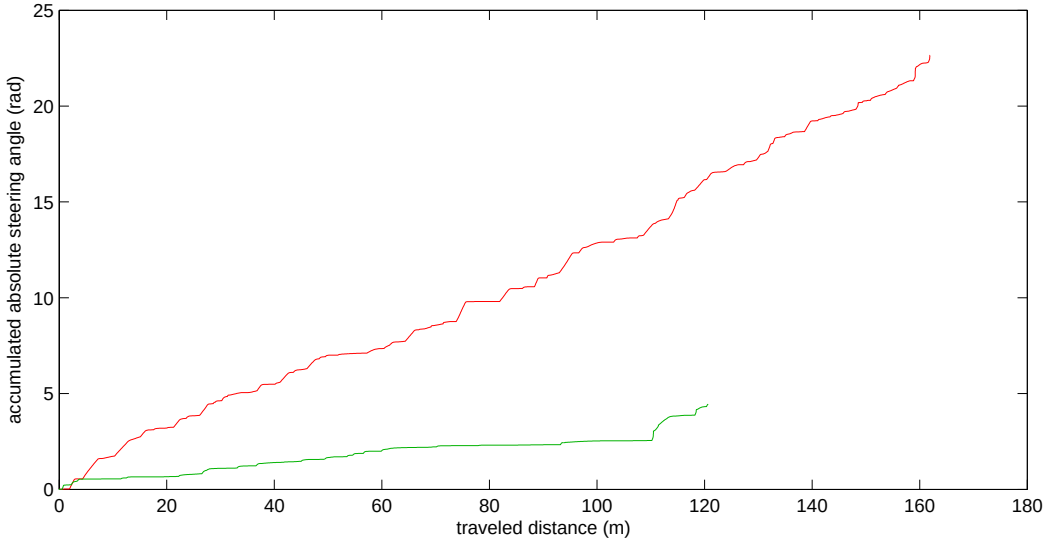
Table 7.7: Results of SPFOA and path corridor-based local planner

planner requires not only less distance to travel, but also less steering. This consequently reduces time cost and energy cost.

Fig. 7.14(a) shows the variation of the steering angle while the robot approaching the goal. The red line represents the result of SPFOA, which requires the robot to frequently apply aggressive (large scale) steering along the way. As it was mentioned in Section 1.2, aggressive steering not only can cost much energy, but also can seriously damage the steering system. On the contrary, the result of the path corridor-based planner (green line) shows a more peaceful steering that allows the robot to move more smoothly. It is also observable that the green line is still a little unstable, however, this is caused by limited accuracy of

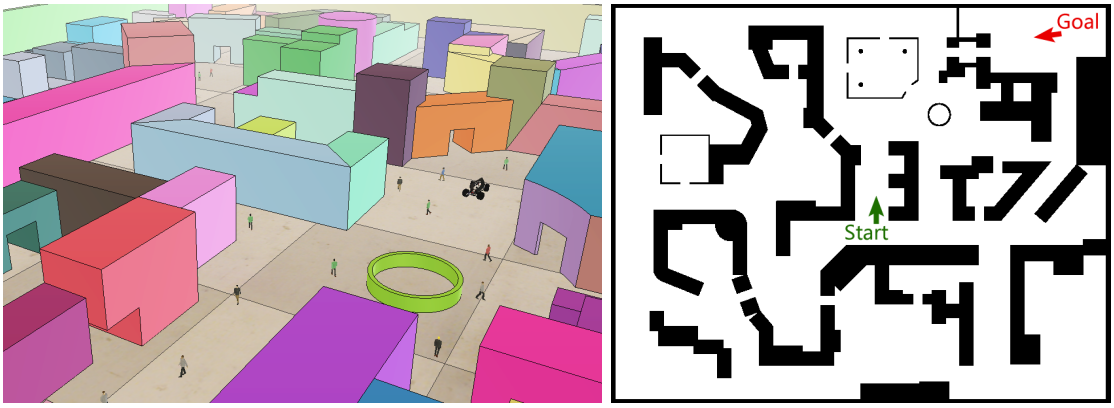


(a)



(b)

Figure 7.14: a) Steering angles of applying SPFOA (red line) and path corridor-based planner (green line) ; b) Accumulated absolute steering angles of SPFOA and path corridor-based planner)



(a) (b)

Figure 7.15: a) Car-like robot simulation in virtual environment; b) Start and goal setup

the controller (Appendix A.4) that is unable to precisely follow the produced local path. However, the topic of the controller is out of the scope of this dissertation.

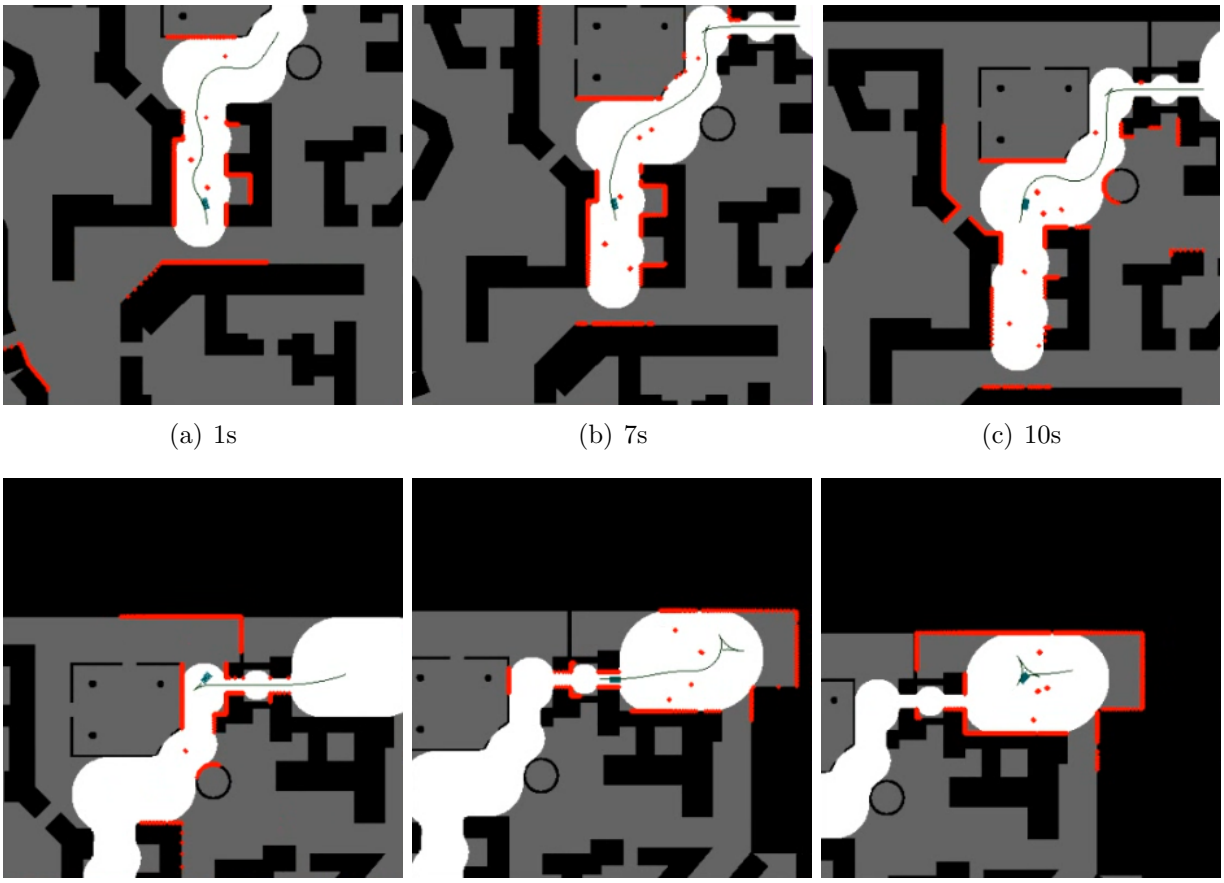
The accumulated absolute steering angle are shown in Fig. 7.14(a), it is observable that the absolute steering angle of SPFOA accumulated much faster than that of the path corridor-based planner. The path corridor-based planner only requires a very small amount of steering. Fig. 7.14(a) further proves the strength of the proposed approach over SPFOA.

7.4.2 Overall Performance under a Clustered Dynamic Environment

[4]This experiment will test the usefulness of corridor-based local path planning in a more sophisticated environment in Fig. 7.15(a). Fig. 7.15(a) shows the simulated shopping center introduced in Appendix A.2). Some the simulated human figures are set to randomly move around. The car-like robot (Appendix A.3) has to avoid all dynamic obstacles while moving in the path corridor and reach the goal. The start and the goal are set as in Fig. 7.15. Since this work is concentrate on mobile robot path planning, the location of the robot is directly provided by the simulator. The localization of the robot is assumed to be done by previous works [123–125].

Fig. 7.16 shows the active window of path corridor-based local path planning at several different simulation time points. The white area represents the path corridor that is produced based on the global path by applying the VHS (Chapter 5). The black objects show the location of the obstacles and the gray area represents the free space outside the path corridor. The robot (the green object) moves on the local path inside the path corridor by following the normative path-based heuristic (Section 6.1.5). The red dots show the locations of the

[4]The video of this experiment is available at `http://youtu.be/mcw-I_2bERA`



(a) 1s (b) 7s (c) 10s

(d) 27s (e) 42s (f) 46s

Figure 7.16: Simulation of path corridor-based path planning

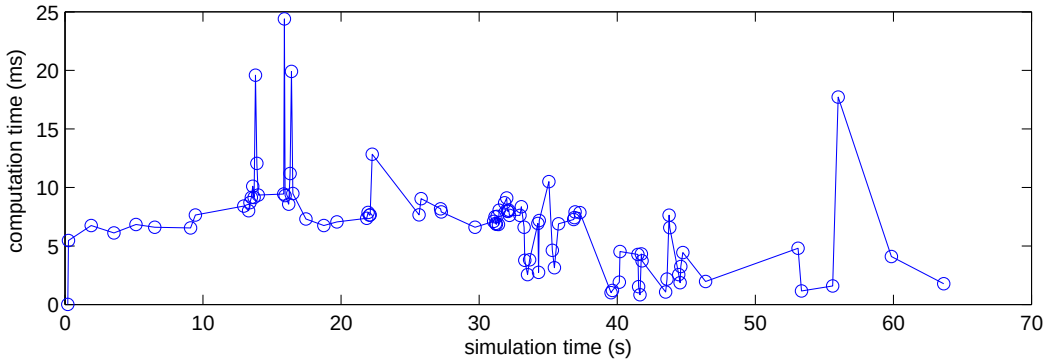


Figure 7.17: Computation time of corridor-based path planning

laser scanner points. The laser scanner points inside the path corridor shows the detected dynamic objects around the robot.

In Fig. 7.16(a) where the robot is initiated from the start point, the local path (shown as the black curve) is produced inside the path corridor and nicely navigated between the dynamic obstacles. While moving on the local path, the robot keeps checking whether or not the local path is still valid. If any dynamic obstacles block the local path or if the controller fails to maintain the robot on the local path, a new local path will be reproduced. Like in the following time sections shown in Fig. 7.16(b) $\sim$ 7.16(e), the local path is frequently reproduced to adapt to the dynamics of the environment while approaching the goal as shown in Fig. 7.16(f).

Fig. 7.17 shows the computation time of the local path planner, the horizontal axis is the simulation time and the vertical axis is the computation time that the planner requires to reproduce the path. The robot takes about 1 minute to reach the goal. From 50 second to 60 second, the path is only reproduced five times, but between 30 second and 40 second, the local path is frequently regenerated (as in Fig. 7.17,). This happens because the local path is not reproduced in a fixed time interval, but only when the current path becomes invalid. Additionally, since the search is required to be done inside the path corridor and the active window, the computation cost is very low. As shown in Fig. 7.17, the computation time is mostly under 10 milliseconds—the maximum computation time is merely 24.4 milliseconds—proving that the proposed corridor-based local path planner is fully capable of on-line computation.

Chapter 8

Conclusion

This dissertation aims to solve several fundamental path planning problems of car-like robots. Robot path planning primarily contains two parts: global path planning and local path planning. Any drawback of these two parts may result in a deterioration in performance of the mobile robot. For instance, local path planning without the support of global path planning may get lost in a large environment. Without feedback from the local planner, global path planning cannot be effectively updated to cope with the environment dynamics. Therefore those parts are not handled separately in this dissertation, but in a unified form as the structure of the system shown in Fig. 1.5. As it will be covered in the following sections, each part is improved for itself and optimized respectively to cope with the other part, to achieve an efficient mobile robot path planning process.

8.1 Achievements

The following sections list the evaluations of the achievements concerning the problems that are expected to be solved within this work. The same section titles are used here for a clear connection with the problems that are listed in the problem statement of the first chapter.

8.1.1 Nonholonomic Constraints

The nonholonomic constraints of the car-like robot are one of the problems that are expected to be solved in this work. The primitive trajectory set is proposed to comply with the nonholonomic constraints of the robot. As the experiments in Section 7.2 and Section 7.3 show, the primitive trajectory set is in a very simple form but, nevertheless, capable of producing various types of maneuvers that do not violate the nonholonomic constraints. As shown by the result in Section 7.3, steering rate and steering frequency are minimized by integrating the steering rate cost in the path cost. This effectively avoids the aggressive steering addressed in Section 1.2 and allows the robot to achieve the goal with very peaceful steering. However, the experiment also shows that reducing the steering rate requires more

computation time. Therefore, a lower steering rate does not necessarily mean a better performance. For real-time local path planning, especially, where the computation must be done in a critical, short time-span, the steering rate cost and the computation time need to be balanced to meet the real-time requirement.

8.1.2 Roadmap

The second problem that needs to be solved is the local minimum, which traps the search and consequently increases the computation time of path planning. As mentioned in Section 1.3, this problem is a consequence of the Euclidean-based heuristic that pulls the search directly towards the goal ignoring the geometric information of the environment. In order to provide a more accurate heuristic estimate, we measure the cost from the current position to the goal along the GVD-based roadmap. The GVD-based roadmap could be considered as the collection of medial lines between the obstacles that connects all the traversable subspaces of the environment. It accurately reflects the geometry of the environment. An improved parallel thinning algorithm is proposed to extract the GVD from the gridmap in Chapter 4. As the experiment in Section 7.1 shows, the proposed thinning algorithm accurately extracts the GVD based on a given gridmap. The produced GVD is always single cell-connected (Section 4.4.2), which increases the accuracy of the cost measured along the GVD and forms a solid foundation for later path planning processes.

8.1.3 Local Minimum of Search-based Path Planning

Several experiments under different environments in Section 7.2 have shown a clear advantage of the proposed Voronoi-based heuristic over the Euclidean-based heuristic. The Voronoi-based heuristic shows the exact behavior as expected in Section 1.3, which guides the search efficiently away from the local minima and, as a result, saves both computation time and memory space. As long as the GVD is extracted from the gridmap, it can be used for multiple queries. Moreover, as Tab. 7.1 shows, the computation time of GVD extraction is very fast. This allows the computation of a single query can also be done within an acceptable time span.

However, providing a more accurate heuristic does not solve the problem of the local minimum once and for all. The GVD only reflects the geometry of the environment, i.e. it only follows the obstacle constraints. As mentioned in Section 5.3, the GVD sometimes goes through narrow corridors that violate the nonholonomics of the car-like robot. Consequently, the search could navigate into corridors that the car-like robot is not capable of. When this happens, the search could still be trapped and form another type of local minimum (NLM) that is newly defined in Section 5.3, which is caused by the obstacle and the nonholonomic constraints. The NLM is the most important challenge of car-like robot path planning that is different from the path planning on other mobile robot platforms. Therefore, another approach is proposed in Section 5.3 to solve this problem, which is able to detect the existence of the NLMs and to navigate the search away from them. As the experiment in Section 7.2.2

shows, the search follows the guidance of the Voronoi-based heuristic, and as soon as it touches the location of the NLM, the search stops and starts to try other possible corridors. The result shows that the proposed approaches can effectively prevent the search from being trapped in the NLM.

8.1.4 Integration of Global and Local Path Planning

Rather than applying the global path directly, the corridor-based local planner transforms the global path into a sequence of corridor sections that define the maximum space along the global path. This is also the major difference of the proposed local path planner from the general approaches introduced in Section 2.5. The experiments in Section 7.4 show the strength of the proposed approach. The corridor-based local planner largely promotes the flexibility of the mobile robot to deal with dynamic obstacles.

Meanwhile, similar to the proposed global path planning, local path planning is guided by the normative path-based heuristic that helps the search navigate through the path corridor. Therefore, the search can effectively move towards the direction led by the global path. This makes the local planner more goal-oriented thus promoting the effectiveness of the search and, consequently, reducing the computation time. The local path is not required to stay with the global path. This preserves both flexibility and effectiveness, and makes the proposed local path planning fully capable of real-time applications.

8.2 Summary

In conclusion, the primary contribution of this work is a new type of heuristic. The proposed heuristic can be combined with various existing search-based path planning algorithms so that the search can smoothly navigate through the space, avoid all kinds of local minima and reach the goal sooner. The proposed approaches have been demonstrated in a simulated environment. The results have shown that the approaches provide high efficiency and reliability, and are highly suitable for real-time applications.

8.3 Future Work

The global path planning algorithm (Chapter 5) is based on a global map created in advance. However, for a real-world application, sometimes the environment is only partially known or even unknown. Therefore, future research should focus on how to incrementally create and update the GVD in real time. Thus, when the environment is partially changed, the Voronoi-based heuristic can also be partially updated immediately. Local path planning (Chapter 6) produces the path based on the situation of the current moment. However, for an environment that includes faster moving dynamic obstacles, the produced path can easily become invalid, necessitating frequent updates of the path. This may lead to unstable results

of the local planner. Therefore, predicting the future movement of the dynamic obstacles and producing the path accordingly are also required.

Appendix A

Simulation and Robot Modeling

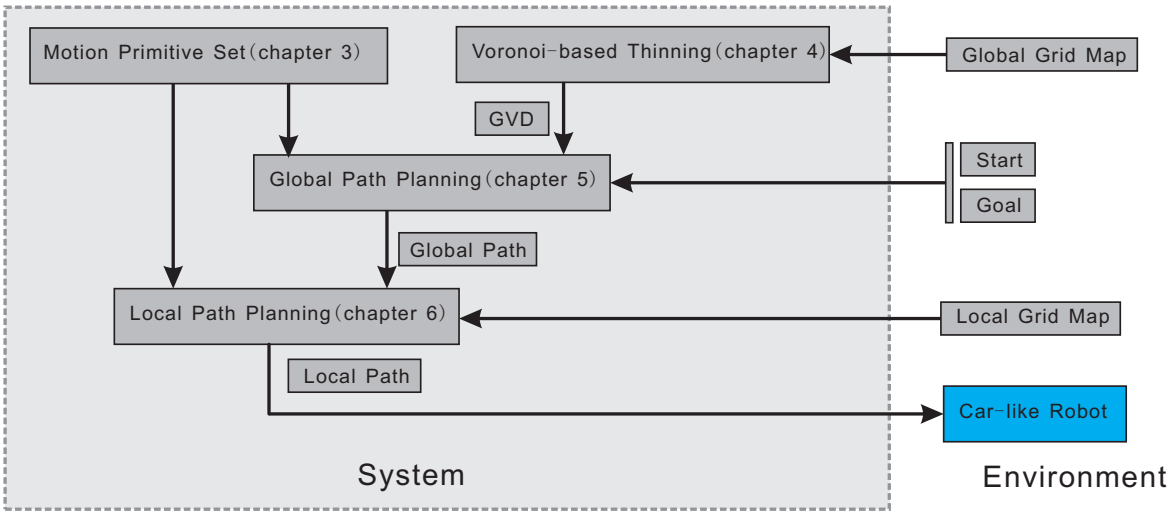


Figure A.1: New Voronoi-based path planning process: simulation and robot modeling

The algorithms are tested under robotic simulator V-REP [126]. The cost of testing the algorithms on a real vehicle is very high since the researchers can be distracted by the construction of the hardware of the experimental platform. Moreover, owing to security issues, the experiments require not only a special place where everything is secured but also a great deal of effort to guarantee that no accident happens. The development of a virtual environment provides an alternative way to test the approaches risk-fee. This work makes use of V-REP, which provides various objects with different physical properties and general sensors like proximity, vision and force sensors. It also provides different types of joints that can be used to create the mechanical movements of a robot.

The car-like robot platform (shown as the highlighted box in Fig. A.1) is a vehicle that executes the result of path planning. The vehicle and the environment dynamics affect the

stability of the controller and the path planner. In order to prove that the system in Fig. A.1 is fully capable of dealing with real-time dynamics, the proposed approaches are tested on a car-like robot, "Manta". In order to integrate more dynamics to the simulation, several approaches to model a car-like robot are proposed, and will be covered later in this chapter.

A.1 Modeling Elements

Joints and pure, simple shapes are the basic elements constructing the dynamics and movements under V-REP.

A.1.1 Joint Types

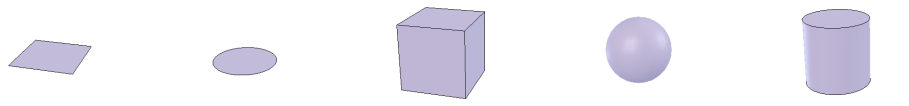
V-REP includes three types of joints (as in Fig. A.2): a revolute joint defining a 1DOF rotational movements between objects; a prismatic joint simulating a 1DOF translational movements between objects; and a spherical joint used to describe a 3DOF rotational movements between objects. In the following sections, the revolute joint is used to create the wheel joints, differential, etc., the prismatic joint is used to create suspensions, and the spherical joint is used as part of the steering mechanism.

(a) Prismatic joint (b) Revolute joint (c) Spherical joint

Figure A.2: Joint types in V-REP

A.1.2 Shapes

Generally speaking, there are two types of shapes under V-REP: pure simple shapes; and simple random shapes. Pure simple shapes (plane, disc, cuboid, cylinder or sphere as in Fig.



(a) Plane (b) Disc (c) Cuboid (d) Sphere (e) Cylinder

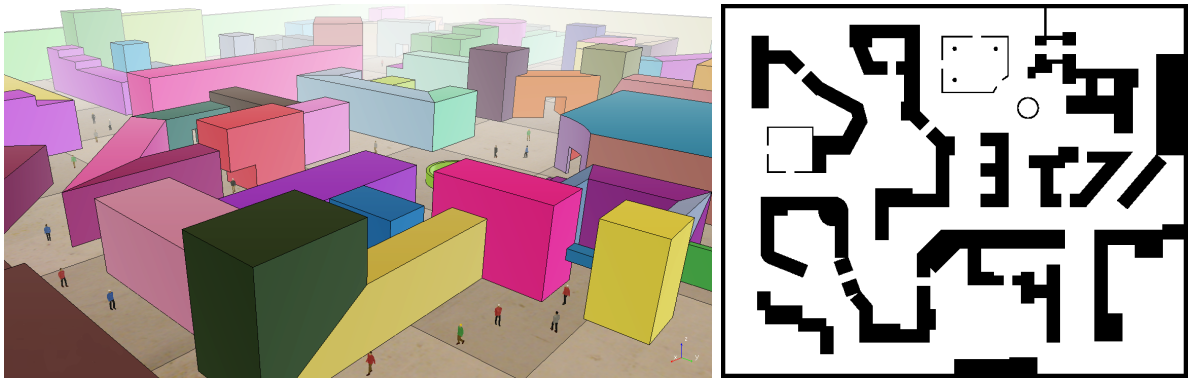
Figure A.3: Pure simple shape

A.3) are the basic units to create the dynamic object. A pure simple shape is best suited for

dynamics and collision response calculation. A simple random shape is used to create the visualized model that can be detected by light-based sensors, or the proximity sensor.

A.2 Simulated Shopping Center

A simulated shopping center is created to test how the algorithms work in a relatively large, dynamic environment. As Fig. A.4(a) shows, there are buildings, and some narrow corridors that normally appear in a residential area. There are also some humanoid models moving around randomly. They are only detectable with proximity sensors. Fig. A.4(b) is the global grid map of the simulated environment, where the car-like robot is expected to move around and provide services.

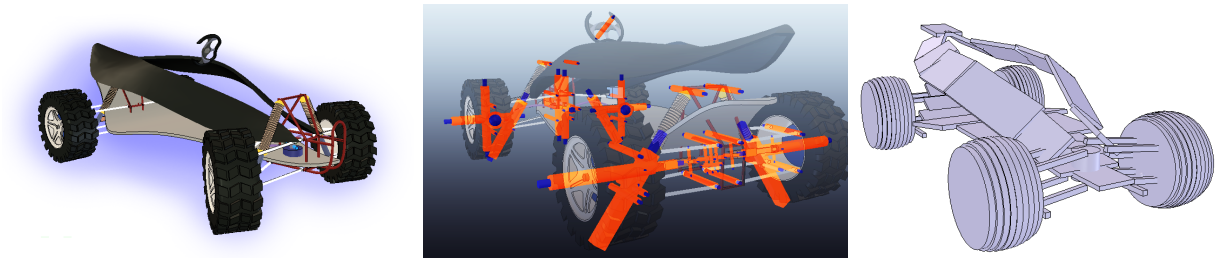


(a) (b)

Figure A.4: a) Simulated Shopping Center; b) Global map of Simulated Shopping Center (750×600)

A.3 Simulated Car-like Robot

Fig. A.5(a) shows car-like robot Manta created under V-REP. Manta is a rear-wheel drive vehicle and is created with general vehicle parts, such as suspension, steering, differential and brakes. Manta is also attached with two motors to motivate steering and linear movements, respectively. A 2D laser scanner and an inertial measurement unit (IMU) are also on board. Fig. A.5(b) shows the body of Manta overlaid with the joints set. There are about 70 joints and each joint can be seen at its corresponding location. Fig. A.5(c) shows the underlying dynamic model of Manta constructed by pure shapes. Manta has been integrated into the model library of V-REP and is available to all interested researchers.

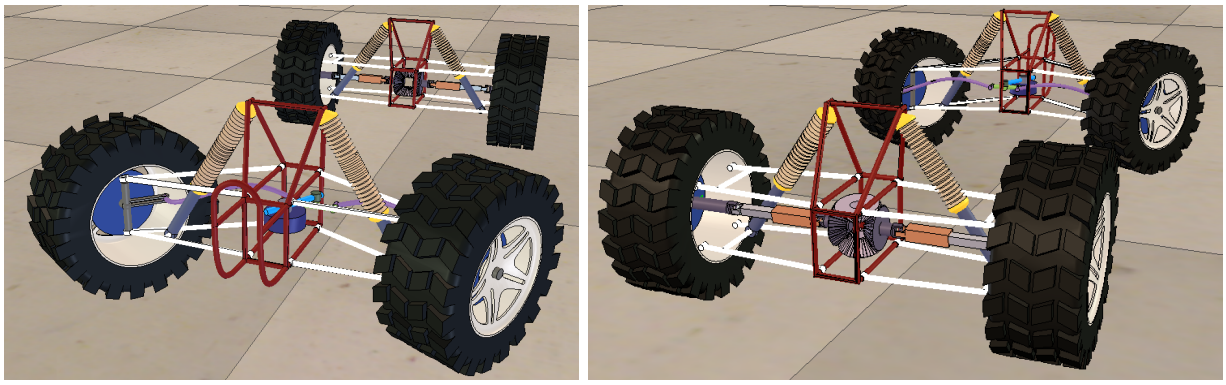


(a) (b) (c)

Figure A.5: a) Full view of Manta; b) Manta's joints set; c) Manta's underlying dynamic model

A.3.1 Suspension

A suspension is used to absorb shocks between the ground and the body of the vehicle, which relatively isolates the body of the vehicle from the unevenness of the ground, and improves both the comfort factor and stability. Manta employs the independent suspension allowing each wheel to move vertically independently from each other.



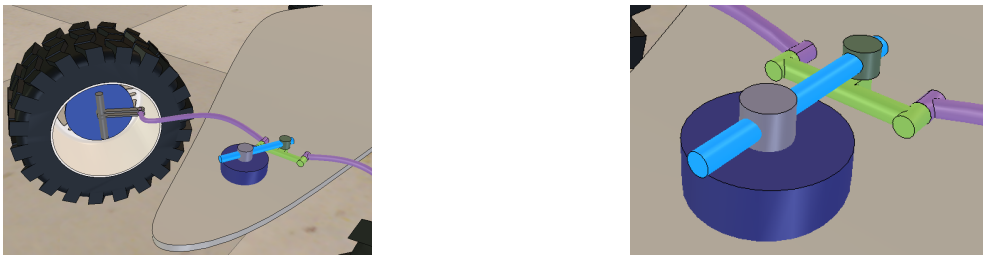
(a) (b)

Figure A.6: Manta's suspension system: a) Front view of Manta's suspension; b) Back view of Manta's suspension

A.3.2 Ackermann steering

As mentioned in Chapter 3, when the vehicle turns into a corner, the Ackermann steering mechanism produces different steering angles for both front wheels—the inside and outside wheels move on the orbits of different radii. The steering system on Manta is implemented mechanically and motivated by a motor in the center of the front wheels (as in Fig. A.7).

The steering angle is limited under $\alpha_{max} = 1/6\pi$ $(30°)$ and the maximum steering rate is set as $\omega_{max} = 7/18\pi$ s^{-1} $(70°$ $s^{-1})$.



(a) (b)

Figure A.7: Manta's steering system

A.3.3 Differential

As Fig. A.8 shows, the cornering angular velocity of inside and outside wheels ω are always the same, but the cornering radius R_{in} and R_{out} are different. Therefore, the linear velocity of the outside wheel ($v_{out} = \omega R_{out}$) is always faster than that of the inside wheel ($v_{in} = \omega R_{in}$). This requires different rolling velocities of both wheels. However, if both wheels are directly connected to the motor, their rolling velocities will always be the same. This poses a conflict between the wheels and the ground, seriously affecting the stability when the vehicle turns. The differential solved such conflict and allows the vehicle to turn peacefully.

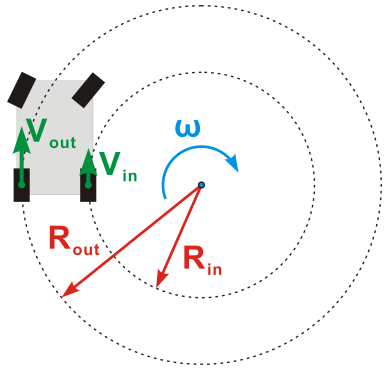


Figure A.8: Velocities of inside and outside wheels while cornering

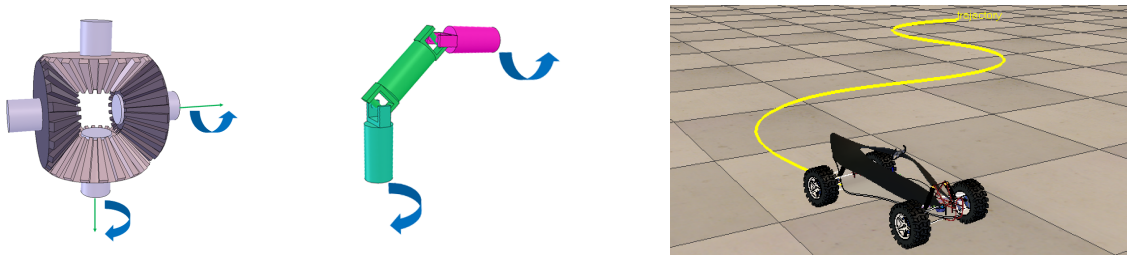
[127, 128] mentioned the ideal conventional differential as (A.1), (A.2) and (A.3). n_{diff} is the gear ratio of the differential. In this case, $n_{diff} = 1$. T_{in} and T_{out} are the torques acting on the inside and outside wheels and ω_{in} and ω_{out} are the associated rolling angular velocities of the inside and outside wheels, respectively. ω_{motor} is the input rolling angular

velocity, and T_{motor} is the torque generated by the motor. It can be seen that T_{in} and T_{out} are always the same, but ω_{in} and ω_{out} are independent of each other.

$$\omega_{motor} = n_{diff}\frac{(\omega_{in} + \omega_{out})}{2} \tag{A.1}$$

$$T_{in} = T_{out} \tag{A.2}$$

$$T_{motor} = n_{diff}(T_{in} + T_{out}) \tag{A.3}$$



(a) (b) (c)

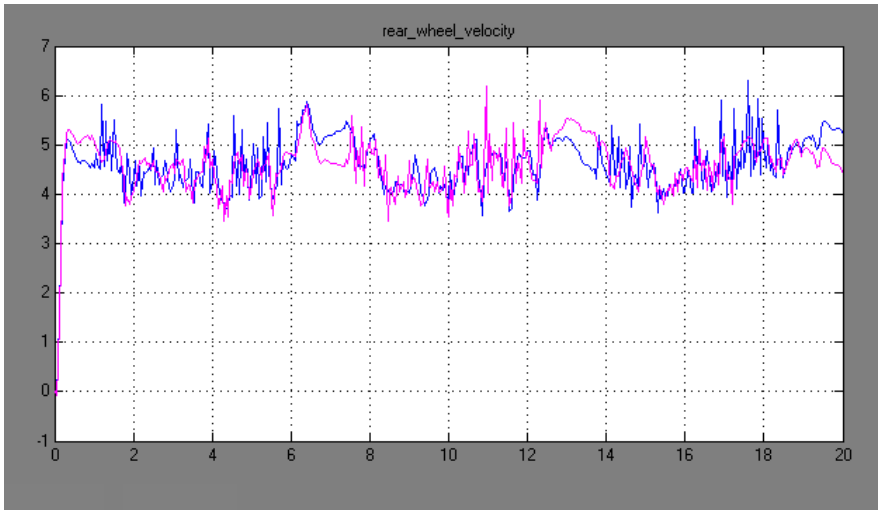
Figure A.9: a) Original differential; b) Substitutive differential; c) Substitutive differential test

The real differential employs gear transmission to implement the differential mechanism. However, gear transmissions cannot be implemented under V-REP directly. Therefore, a substitutive differential is proposed in this section. Gear transmission of the differential is simply a mechanism that transmits angular velocity and torque to a different direction (as in Fig. A.9). Such a mechanism can be replaced with several universal joints that are also able to transmit angular velocity and torque to a different direction.

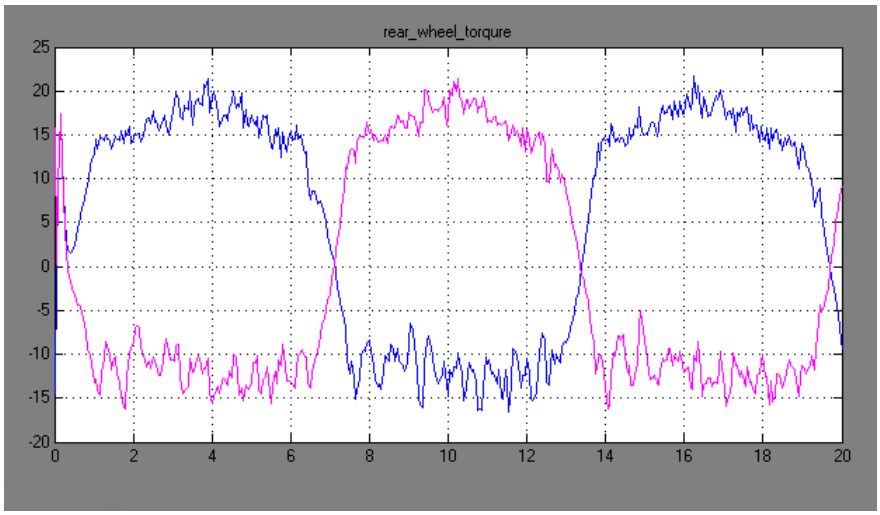
Fig. A.10 shows the test without the differential, whereas Fig. A.11 is the test using the substitutive differential implemented with universal joints in which the outputs of both rear wheels are shown in blue and pink lines, respectively. In both tests, robot Manta runs with the same sinusoidal steering input under the same linear velocity [as in Fig. A.9(c)].

In Fig. A.10, the result shows the test without the differential—both rear wheels are connected directly to the motor. The rolling angular velocities [Fig. A.10(a)] and torques [Fig. A.10(b)] of both rear wheels are very unstable because of the conflict between the inside wheel and the outside wheel. As a result, the torques acting on both wheels reach very high level making the vehicle very unstable and significantly disturbing the dynamics of the system.

Fig. A.11 is the result of using the substitutive differential. It can be seen in Fig. A.11(a) that the movements of both wheels become more peaceful with the substitutive differential. The torques shown in Fig. A.11(b) are evenly distributed between both wheels and stay at a

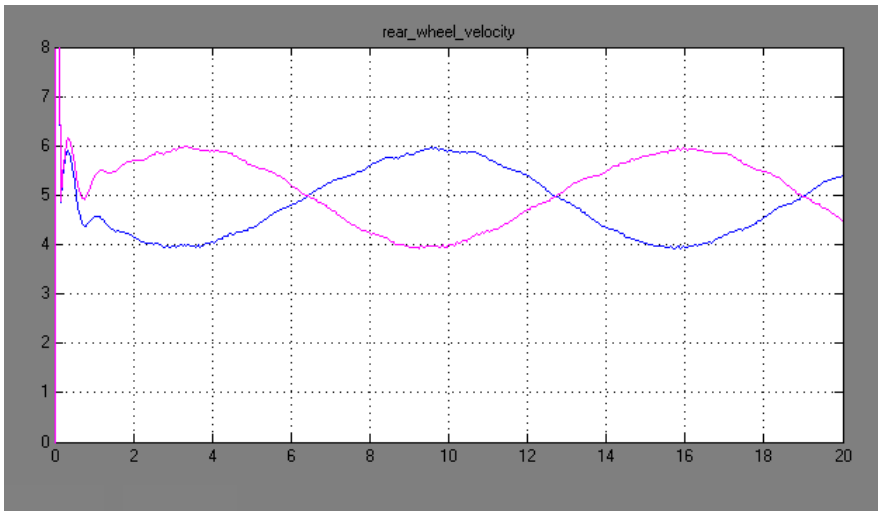


(a)

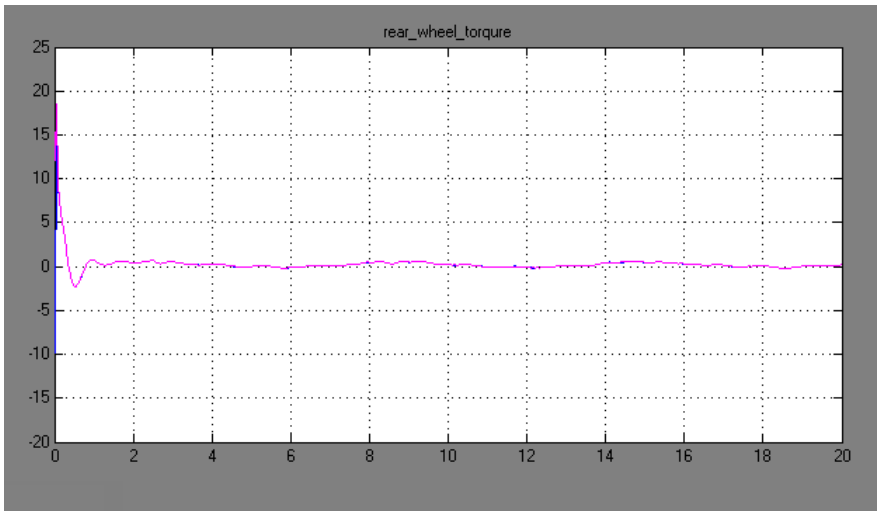


(b)

Figure A.10: Test without deferential: a) Rolling angular velocities of rear wheels; b) Torques acted on the rear wheels



(a)



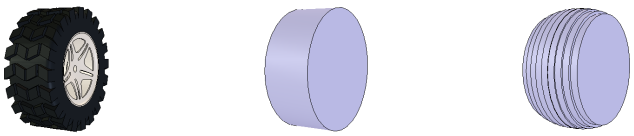
(b)

Figure A.11: Test with substitutive deferential: a) Rolling angular velocities of rear wheels; b) Torques act on rear wheels

very low level. With the help of the substitutive differential, the movements of both wheels become independent relative to each other, and the conflict between them disappears. The rolling angular velocities of both wheels vary alternately as the wheels alternate to be the inside and outside wheels as shown in Fig. A.11(a). The result shows that the substitutive differential serves in the same way of a real differential defined by (A.1), (A.2) and (A.3).

A.3.4 Wheels

Fig. A.12(a) is the visualized tire model. Fig. A.12(b) is a general underlying cylinder tire model. However, since such a cylinder tire model is a rigid object, it prevents itself from tilting sideways as the force acts horizontally. This affects the dynamic performance of the vehicle since the flexibility of the wheels is constrained when the vehicle steers.



(a) (b) (c)

Figure A.12: a) Tire view; 2) Cylinder tire model; 3) Improved tire model

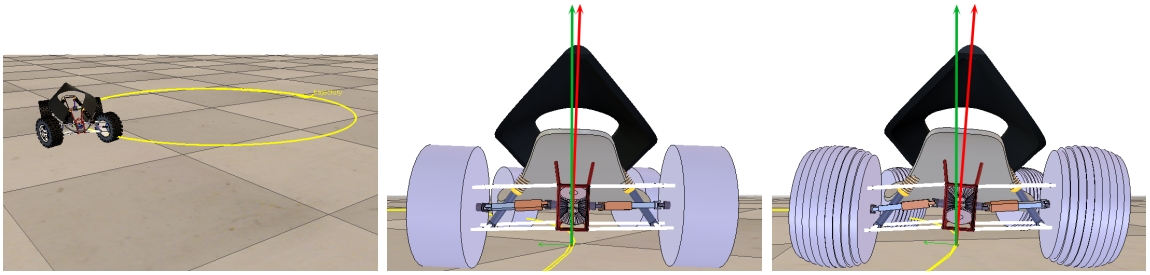
The improved underlying tire model in Fig. A.12(c) consists of a sequence of very thin discs with varied radius R_i defined by (A.4), where R_{wheel} is the radius of the wheel. d_i is the lateral distance between the ith disc and the center of tire. The improved tire model forms a circular surface, and allows the tires to tilt sideways more flexibly.

$$R_i = \sqrt{R_{wheel}^2 - d_i^2} \tag{A.4}$$

Fig. A.13 is the test of applying different tire models. Manta runs on a circular path with both wheel models and this generates a centrifugal force applied on the wheels horizontally. The green arrow shows the reference direction perpendicular to the ground, whereas the red arrow is the central line attached to Manta's body. Compared with the simple cylinder tire model [Fig. A.13(b)], Manta with the improved tire model [Fig. A.13(c)] preserves more dynamics.

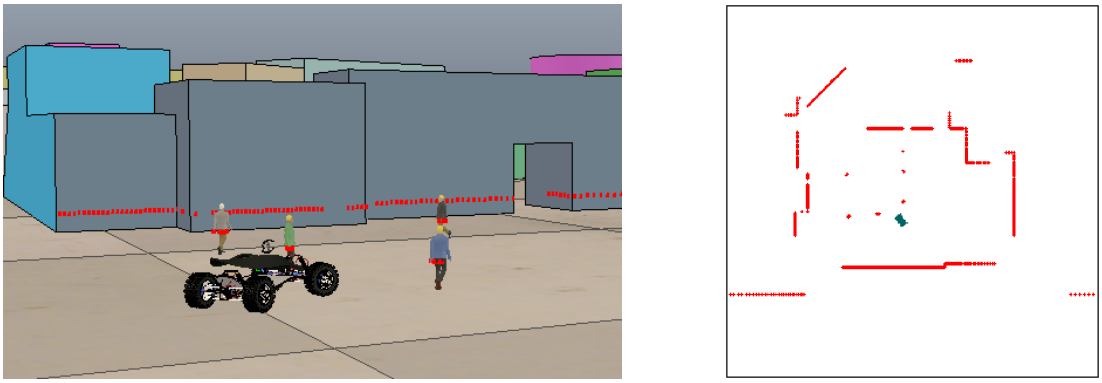
A.3.5 Sensors

As mentioned above, Manta is also attached with a 2D, 360° laser scanner as shown in Fig. A.14. Since this work plans to concentrate on the path planning problem, it is assumed that the localization of the robot has been done by other processes. Therefore, the accurate position and orientation are directly provided by the simulator.



(a) (b) (c)

Figure A.13: Tire model test



(a) (b)

Figure A.14: a) Virtual environment with dynamic objects; b) Laserscan points

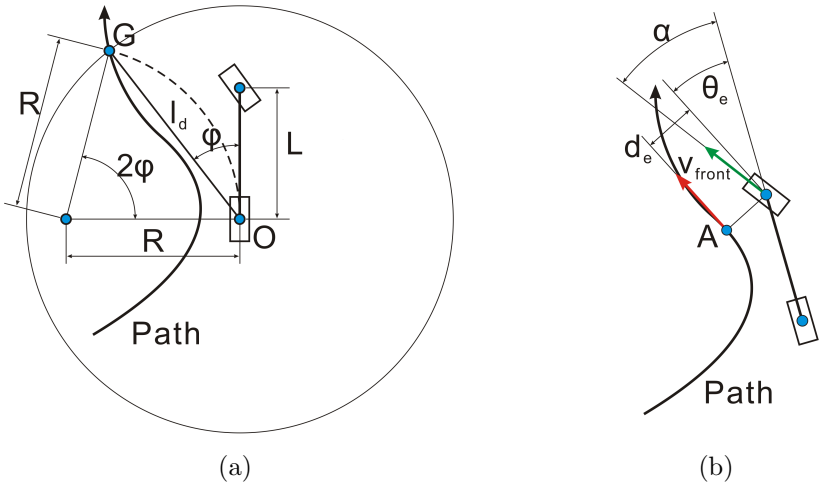
A.4 Controller

In order to execute the result of the global and local path planning proposed in the previous chapters, a controller needs to be used to move the car-like robot on the path. Pure pursuit [129] is one of the widely used path following controllers, and is simple and easy to apply. Pure pursuit is based on the kinematic model of a car-like robot. The object with two connected rectangles [as in Fig. A.15(a)] represents a simplified bicycle model of the car-like robot mentioned in Section 2 [Fig. 2.10(a)], where the black curve with an arrow is the path that the robot is supposed to follow. G is the reference point on the path, with distance l_d to center O of the rear wheels. φ is the angle between GO and the orientation of the vehicle. With steering angle α calculated based on (A.5),

$$\alpha = tan^{-1}(\frac{2Lsin(\varphi)}{l_d}) \tag{A.5}$$

the vehicle moves towards reference point G along the dashed curve, where R is the radius of the curve. If l_d is too small, the system becomes very sensitive to the error since the controller

applies a large input to rectify the error, which can make the vehicle oscillate along the path. With a large value of l_d, the system is much more stable. However, it becomes less sensitive and may generate a large error. [130] suggests setting $l_d = \kappa_{pure} v$, where v is the current linear velocity of the robot and κ_{pure} is the coefficient to tune the effect of v.



(a) (b)

Figure A.15: a) Pure pursuit; b) Stanley method

The Stanley method [131] is another controller applied to the autonomous vehicle Stanley that won the DARPA Grand Challenge. Unlike pure pursuit, the reference point of this method is based on the center of the front wheels [as in Fig. A.15(b),]. The Stanley method consists of two terms [as in (A.6)].

$$\alpha = \theta_e + tan^{-1}(\frac{\kappa_{stan} d_e}{v_{front}}) \tag{A.6}$$

The first term represents angle θ_e between the orientation of the vehicle and the reference direction of point A (shown as the arrowed red line) on the path, where point A is the reference point of the front wheel on the path. Distance error d_e between the vehicle and the reference point is then transformed into the second term of (A.6), where v_{front} is the moving velocity of the front wheel (shown as the arrowed green line). The combination of both terms reduces both the orientation and the position error, where κ_{stan} is used to tune the effect of the position error. However, Stanley is more suitable for a forward drive, because its backward drive produces a relatively large error.

The kinematic controller [132] is based on the chained form described in [133, 134]; this method can track the path with a very small error under a low velocity. However, its robustness worsens as the speed increases. This feature makes it more useful for parking or navigating through winding narrow corridors, where precise control over the vehicle is required.

The controllers are based on the assumption that there is no lateral movement of the wheels. For low to medium speed, this assumption holds since the lateral displacement is small enough to be ignored. However, as the speed increases, the kinematic model can no longer be applied. In such case, the dynamic model considering the lateral displacement of the wheels should be used [135–138]. However, for most off-road environments, the vehicle is not expected to move at very high speeds. Furthermore, as long as the position error goes beyond the threshold, it can be rectified by regenerating a new path based on the current state of the vehicle. Therefore, a controller of very high accuracy is not really necessary. In this work, pure pursuit is applied as the path following controller for its simplicity in the forward and backward drives.

A.5 Summary

A simulated vehicle and an environment under simulator V-REP have been created to test the proposed approaches of this work. The simulation environment may not be realistic enough and some of physical parts of the vehicle are simplified. For instance, the friction is only implemented with the simple coulomb friction model [139]. However, the simulated car-like robot still gives a very good performance and greatly decreases the cost of the experiment compared to a real robotic platform.

Bibliography

[1] M. Hentschel. *Langzeitnavigation mobiler Roboter in teilstrukturierten Umgebungen auf Basis eines raum- und zeitvarianten Umgebungsmodells*. PhD thesis, Gottfried Wilhelm Leibniz Universität Hannover, 2012.

[2] S. M. LaValle and J. J. Kuffner Jr. Randomized kinodynamic planning, 1999.

[3] S. Quinlan and O. Khatib. Elastic bands. connecting path planning and control. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 802–807, 1993.

[4] J. Markoff. Google cars drive themselves, in traffic. *The New York Times*, October 2010. `http://www.nytimes.com/2010/10/10/science/10google.html?_r=0`.

[5] A. Fisher. Inside google's quest to popularize self-driving cars. *Popular Science*, September 2013. `http://www.popsci.com/cars/article/2013-09/google-self-driving-car`.

[6] The self-driving car logs more miles on new wheels. *googleblog*. `http://googleblog.blogspot.hu/2012/08/the-self-driving-car-logs-more-miles-on.html`.

[7] S. Thrun. What were driving at. *The Official Google Blog*, October 2010. `http://googleblog.blogspot.de/2010/10/what-were-driving-at.html`.

[8] A. Kelly and M. Pivtoraiko. Generating near minimal spanning control sets for constrained motion planning in discrete state spaces. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3231 – 3237, August 2005.

[9] A. Scheuer and T. Fraichard. Planning continuous-curvature paths for car-like robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 1304–1311, November.

[10] K. Popp and We. Schiehlen. *Ground Vehicle Dynamics*. B.G.Teubner, 2010.

[11] B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33, 2001.

[12] P. Chakrabarti, S. Ghosh, and S. DeSarkar. Admissiblility of AO* when heuristics overestimate. *Artificial Intelligence*, 34:97–113, 1988.

[13] S. Edelkamp. Planning with pattern databaseds. In *European Conference on Planning*, 2001.

[14] R. Korf. Linear-space best-first search. *Artificial Intelligence*, 62:41–78, 1993.

[15] M. Likhachev, G. Gordon, and S. Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. In *Advances in Neural Information Processing Systems*. MIT press.

[16] R. Zhou and E. Hansen. Multiple sequence alignment using A*. In *National Conference on Artificial Intelligence*, 2002.

[17] R. A. Knepper and A. Kelly. High performance state lattice planning using heuristic look-up tables. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3375 – 3380, October 2006.

[18] B. Mirtich and J. Canny. Using skeletons for nonholonomic path planning among obstacles, 1992.

[19] D. A. L. García and F. Gomez-Bravo. Vodec: A fast voronoi algorithm for car-like robot path planning in dynamic scenarios. *Robotica*, 30(7):1189–1201, 2012.

[20] M. Foskey, M. Garber, M. C. Lin, and D. Manocha. A voronoi-based hybrid motion planner for rigid bodies. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 55–60, 2001.

[21] O. Takahashi and R. J. Schilling. Motion planning in a plane using generalized voronoi diagrams. In *IEEE Transactions on Robotics and Automation*, pages 143–150, April 1989.

[22] M. Seder, K. Macek, and I. Petrovic. An integrated approach to real-time mobile robot control in partially known indoor environments. In *Proceedings of 31st Annual Conference of IEEE Industrial Electronics Society. IECON.*, page 6 pp., 2005.

[23] L. Lapierre, R. Zapata, and P. Lepinay. Combined path-following and obstacle avoidance control of a wheeled robot. *The International Journal of Robotics Research*, 26(4):361–375, 2007.

[24] B. T. Bertka. An Introduction to Bezier Curves, B-Splines, and Tensor Product Surfaces with History and Applications. 2008.

[25] S. Thrun and A. Buecken. Integrating grid-based and topological maps for mobile robot navigation. volume 2, pages 944–950, 1996.

[26] Y. Hu, J. Gong, Y. Jiang, L. Liu, G. Xiong, and H. Chen. Hybrid map-based navigation method for unmanned ground vehicle in urban scenario. *Remote Sensing*, 5(8):3662–3680, 2013.

[27] S. Jia, H. Shen, X. Li, and K. Wang. Research on autonomous robots' environment exploration based on hybrid environment model. *Gaojishu Tongxin/Chinese High Technology Letters*, 23(7):756–761, 2013.

[28] M. Nitsche, P. De Cristóforis, M. Kulich, and K. Košnar. Hybrid mapping for autonomous mobile robot exploration. In *The 6th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS'2011*, volume 1, pages 299–304, 2011.

[29] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. Cambridge, Mass: MIT Press, 2005.

[30] E. Einhorn, C. Schröter, and H. Gross. Finding the adequate resolution for grid mapping - cell sizes locally adapting on-the-fly. In *IEEE International Conference on Robotics and Automation*, pages 1843–1848, 2011.

[31] K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard. Octomap: A probabilistic, flexible, and compact 3d map representation for robotic systems. In *IEEE International Conference on Robotics and Automation workshop*, 2010.

[32] G. K. Kraetzschmar, G. P. Gassull, and K. Uhl. Probabilistic quadtrees for variable-resolution mapping of large environments. In *The 5th IFAC/EURON*, 2004.

[33] J. Vörös. Low-cost implementation of distance maps for path planning using matrix quadtrees and octrees. *Robotics and Computer-Integrated Manufacturing*, 17(6):447–459, 2001.

[34] F. Wang, C. Zhou, and H. de Garis. A simple robot paths planning based on quadtree. In *IEEE International Conference on Progress in Informatics and Computing, PIC 2010*, volume 1, pages 1–4, 2010.

[35] M. A. Movafaghpour and E. Masehian. Poly line map extraction in sensor-based mobile robot navigation using a consecutive clustering algorithm. *Robotics and Autonomous Systems*, 60(8):1078 – 1092, 2012.

[36] C. Fernndez, V. M., B. Curto, and J. A. Vicente. Clustering and line detection in laser range measurements. *Robotics and Autonomous Systems*, 58(5):720–726, 2010.

[37] Y. Zhao and X. Chen. Prediction-based geometric feature extraction for 2d laser scanner. *Robotics and Autonomous Systems*, 59(6):402 – 409, 2011.

[38] T. Lozano-Pérez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, October 1979.

[39] G. P. Lorenzetto, A. Datta, and R. Thomas. A fast trapezoidation technique for planar polygons. *Computers and Graphics*, 26(2):281–289, 2002.

[40] B. Chazelle and J. Incerpi. Triangulation and shape-complexity. *ACM Transactions on Graphics*, 3(2):135–152, April 1984.

[41] M. Bern and D. Eppstein. Mesh generation and optimal triangulation, 1992.

[42] L. Kavraki, P. Svestka, J. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. In *IEEE International conference on Robotics and Automation (ICRA)*, pages 566–580, 1996.

[43] G. L. Dirichlet. Über die reduktion der positiven quadratischen formen mit drei unbestimmten ganzen zahlen. *J. Reine Angew. Math.*, 1850.

[44] G. M. Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. deuxième mémoire: Recherches sur les parallélloèdres primitifs. *J. Reine Angew. Math.*, 1908.

[45] S. Fortune. A sweepline algorithm for voronoi diagrams. *Algorithmica*, 2(1-4):153–174, 1987.

[46] H. Inagaki, K. Sugihara, and N. Sugie. Numerically robust incremental algorithm for constructing three-dimensional voronoi diagrams. In *The 4th Canadian Conference on Computational Geometry*, pages 334–339, 1992.

[47] K. Sugihara and M.Iri. A robust topology-oriented incremental algorithm for voronoi diagrams. *International Journal of Computational Geometry and Applications*, 1994.

[48] H. Choset and J. Burdick. Sensor based exploration: The hierarchical generalized voronoi graph, 2000.

[49] D. T. Lee. Medial axis transformation of a planar shape. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, July 1982.

[50] M. Held. Voronoi diagrams and offset curves of curvilinear polygons. *Computer-Aided Design*, 30(4):287 – 300, 1998.

[51] V. Milenkovic. Robust polygon modelling. *Computer-Aided Design*, 25(9):546 – 566, 1993.

[52] K. Hoff, T. Culver, J. Keyser, M. Lin, and D. Manocha. Fast computation of generalized voronoi diagrams using graphics hardware. In *The 26th International Conference on Computer Graphics and Interactive Techniques*, pages 277–286, 1999.

[53] T. Y. Zhang and C. Y. Suen. A fast parallel algorithms for thinning digital patterns. *Communications of the ACM*, 3(27):236–239, March 1984.

[54] B. Jang and R. T. Chin. Analysis of thinning algorithms using mathematical morphology. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(6):541–551, 1990.

[55] F. Leymarie and M. D. Levine. Simulating the grassfire transform using an active contour model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(1):56–75, 1992.

[56] R. Kimmel, D. Shaked, N. Kiryati, and A. M. Bruckstein. Skeletonization via distance maps and level sets. *Computer Vision and Image Understanding*, 62(3):382–391, 1995.

[57] L. F. Costa. Enhanced multiscale skeletons. *Real-Time Imaging*, 9(5):315 – 319, 2003.

[58] H. Liu, Z. Wu, D. F. Hsu, B. S. Peterson, and D. Xu. On the generation and pruning of skeletons using generalized voronoi diagrams. *Pattern Recognition Letters*, 33(16):2113 – 2119, 2012.

[59] B. Lau, C. Sprunk, and W. Burgard. Improved updating of euclidean distance maps and voronoi diagrams. In *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems*, pages 281–286, 2010.

[60] R. Lotufo and F. Zampirolli. Fast multi-dimensional parallel euclidean distance transform basedon mathematical morphology. In *SIBGRAPI, XIV Brazilian Symposium on ComputerGraphics and Image Processing. IEEE Computer Society*, page 100105, 2001.

[61] C. R. Maurer, R. Qi, and V. Raghavan. A linear time algorithm for computing exact euclidean distance transforms of binary images in arbitrary dimensions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(2):265–270, 2003.

[62] A. Meijster, J. B. T. M. Roerdink, and W. H. Hesselink. A general algorithm for computing distance transforms in linear time. In *The 5th International Conference on Mathematical Morphology and its Applications to Image and Signal Processing*, 2000.

[63] Y. S. Frank and Y. Wu. Fast euclidean distance transformation in two scans using a 3x3 neighborhood. *Computer Vision and Image Understanding*, 93(2):195 – 205, 2004.

[64] S. M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, 1998.

[65] J. J. Kuffner Jr. and S. M. LaValle. RRT-connect: an efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 995–1001, 2000.

[66] S. M. Lavalle and J. J. Kuffner Jr. Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and Computational Robotics: New Directions*, pages 293–308, 2000.

[67] P. Cheng and S. M. LaValle. Reducing metric sensitivity in randomized trajectory design. In *IEEE International Conference on Intelligent Robots and Systems*, volume 1, pages 43–48, 2001.

[68] M. Kalisiak and M. Van De Panne. Rrt-blossom: Rrt with a local flood-fill behavior. In *IEEE International Conference on Robotics and Automation*, volume 2006, pages 1237–1242, 2006.

[69] M. Strandberg. Augmenting rrt-planners with local trees. In *IEEE International Conference on Robotics and Automation*, volume 2004, pages 3258–3262, 2004.

[70] L. Kavraki and J. Latombe. Randomized preprocessing of configuration space for fast path planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2138–2145, 1994.

[71] N. M. Amato and Y. Wu. A randomized roadmap method for path and manipulation planning. In *IEEE International conference on Robotics and Automation (ICRA)*, pages 113–120, 1996.

[72] J. Barraquand, L. Kavraki, J. Latombe, T. Li, R. M., and P. Raghavan. A random sampling scheme for path planning. *International Journal of Robotics Research*, 16:759–774, 1996.

[73] L. E. Kavraki. *Random networks in configuration space for fast path planning*. PhD thesis, Stanford, CA, USA, 1995. UMI Order No. GAX95-16854.

[74] M. H. Overmars. A random approach to motion planning. Technical report, 1992.

[75] J. Corts, T. Simon, and J. P. Laumond. A random loop generator for planning the motions of closed kinematic chains using prm methods. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 2141–2146, 2002.

[76] D. Xie and N. M. Amato. A kinematics-based probabilistic roadmap method for high dof closed chain systems. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 2004, pages 473–478, 2004.

[77] C. Holleman, L. E. Kavraki, and J. Warren. Planning paths for a flexible surface patch. In *IEEE International conference on Robotics and Automation (ICRA)*, pages 21–26, 1998.

[78] F. Lamiraux and L. E. Kavraki. Planning paths for elastic objects under manipulation constraints. *International Journal of Robotics Research*, 20:188–208, 2001.

[79] G. Snchez and J. Latombe. A single-query bi-directional probabilistic roadmap planner with lazy collision checking, 2001.

[80] T. Simon, J. Cortes, A. Sahbani, and J. P. Laumond. A manipulation planner for pick and place operations under continuous grasps and placements. In *IEEE International conference on Robotics and Automation (ICRA)*, pages 2022–2027, 2002.

[81] P. Svestka and M. H. Overmars. Coordinated path planning for multiple robots, 1998.

[82] J. Barraquand, L. Kavraki, J. Latombe, R. Motwani, T. Li, and P. Raghavan. A random sampling scheme for path planning. *International Journal of Robotics Research*, 16(6):759–774, December 1997.

[83] L. Lulu and A. Elnagar. A comparative study between visibility-based roadmap path planning algorithms. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3700–3705, 2005.

[84] D. Hsu, J. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *IEEE International Conference on Robotics and Automation*, volume 3, pages 2719–2726, 1997.

[85] C. Nissoux, T. Simeon, and J. Laumond. Visibility based probabilistic roadmaps. In *IEEE International Conference on Intelligent Robots and Systems*, volume 3, 1999.

[86] Y. Yang and O. Brock. Adapting the sampling distribution in PRM planners based on an approximated medial axis. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 5, pages 4405–4410, April 2004.

[87] J. Lien, S.L. Thomas, and N.M. Amato. A general framework for sampling on the medial axis of the free space. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 3, pages 4439–4444 vol.3, Sept 2003.

[88] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269271, 1959.

[89] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. 4(2):100107, 1968.

[90] A. Stentz. The focussed D* algorithm for real time replanning. In *International Joint Conference on Artificial Intelligence*, 1995.

[91] S. Koenig and M. Likhachev. Improved fast replanning for robot navigation in unknown terrain. In *IEEE International Conference on Robotics and Automation*, 2002.

[92] D. Ferguson and A. Stentz. The delayed D* algorithm for efficient path replanning. In *IEEE International Conference on Robotics and Automation*, 2005.

[93] M. Likhachev and D. Ferguson. Planning long dynamically feasible maneuvers for autonomous vehicles. *The International Journal of Robotics Research*, 28(8):933–945, 2009.

[94] E. Mazer, J. M. Ahuactzin, E. Talbi, and P. Bessiere. The ariadne's clew algorithm, 1996.

[95] D. Knowles and M. R. Murray. Real time continuous curvature path planner for an autonomous vehicle in an urban environment, 2006.

[96] Y. Kanayama and B. I. Hartman. Smooth local path planning for autonomous vehicles. In *IEEE International Conference on Robotics and Automation*, volume 3, pages 1265–1270, May 1989.

[97] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *IEEE Conference on Robotics and Automation*, volume 2, pages 500–505, March 1985.

[98] J. Guldner, V. I. Utkin, and Bauer R. Mobile robots in complex environments: a three-layered hierarchical path control system. In *IEEE/RSJ/GI International Conference on Intelligent Robots and Systems*, volume 3, pages 1891–1898, 1994.

[99] J. Guldner, V. I. Utkin, and R. Bauer. A three-layered hierarchical path control system for mobile robots: Algorithms and experiments. *Robotics and Autonomous Systems*, 14(2-3):133–147, 1995.

[100] E. Shi, T. Cai, C. He, and J. Guo. Study of the new method for improving artifical potential field in mobile robot obstacle avoidance. In *Proceedings of Proceedings of 2007 IEEE International Conference on Automation and Logistics*, pages 282–286, Aug 2007.

[101] J. Borenstein and Y. Koren. The vector field histogram - fast obstacle avoidance for mobile robots. *IEEE Journal of Robotics and Automation*, 7:278–288, 1991.

[102] H. Kim, K. Lee, K. Ryu, and M. Park. Navigation control of mobile robot using distance profile histogram. In *IEEE International Conference on Intelligent Robots and Systems*, volume 2, pages 949–956, 1996.

[103] I. Ulrich and Borenstein J. VFH+: Reliable obstacle avoidance for fast mobile robots. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1572–1577, 1998.

[104] B. You, J. Qiu, and D. Li. A novel obstacle avoidance method for low-cost household mobile robot. In *Proceedings of IEEE International Conference on Automation and Logistics, 2008. ICAL 2008.*, pages 111–116, Sept 2008.

[105] D. An and H. Wang. VPH: a new laser radar based obstacle avoidance method for intelligent mobile robots. In *Proceedings of Fifth World Congress on Intelligent Control and Automation, 2004. WCICA 2004.*, volume 5, pages 4681–4685 Vol.5, June 2004.

[106] I. Ulrich and J. Borenstein. VFH*: Local obstacle avoidance with look-ahead verification, 2000.

[107] S. Quinlan and O. Khatib. Towards real-time execution of motion tasks, 1993.

[108] O. Brock and O.Khatib. Executing motion plans for robots with many degrees of freedom in dynamic environments. In *International Conference on Robotics and Automation*, pages 1–6, 1998.

[109] B. Graf, J. M. H. Wandosell, and C. Schaeffer. Flexible path planning for nonholonomic mobile robots. In *The 4th European Workshop on Advanced Mobile Robots (EUROBOT)*, page 199206, 2001.

[110] R. Geraerts and M. H. Overmars. The corridor map method: A general framework for real-time high-quality path planning. *Computer Animation and Virtual Worlds*, 18:107–119, 2007.

[111] J. Hilgert, K. Hirsch, T. Bertram, and M. Hiller. Emergency path planning for autonomous vehicles using elastic band theory. In *Proceedings of Proceedings. 2003 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, volume 2, pages 1390–1395 vol.2, July 2003.

[112] T. Sattel and T. Brandt. Ground vehicle guidance along collision-free trajectories using elastic bands. In *American Control Conference*, volume 7, pages 4991–4996, 2005.

[113] L. Lapierre, R. Zapata, and P. Lepinay. Simulatneous path following and obstacle avoidance control of a unicycle-type robot. In *Proceedings of 2007 IEEE International Conference on Robotics and Automation*, pages 2617–2622.

[114] M. Seder and I. Petrovic. Dynamic window based approach to mobile robot motion control in the presence of moving obstacles. In *Proceedings of 2007 IEEE International Conference on Robotics and Automation*, pages 1986–1991.

[115] H. Nooraliei and S. A. Mostafa. Robot path planning using wave expansion approach virtual target. In *Proceedings of International Conference on Computer Technology and Development, 2009. ICCTD '09.*, volume 1, pages 169–172.

[116] M. Hentschel, O. Wulf, and B. Wagner. A hybrid feedback controller for car-like robots - combining reactive obstacle avoidance and global replanning. In *The 3rd International Conference on Informatics in Control, Automation and Robotics*, pages 1–5, August 2006.

[117] G. E. Farin, J. Hoschek, and M. Kim. *Handbook of Computer Aided Geometric Design*. Elsevier, 2002.

[118] J. King and M. Likhachev. Efficient cost computation in cost map planning for non-circular robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3924–3930, Saint Louis, Missouri, USA, October 2009.

[119] D. Eberly. Skeletonization of 2d binary images, June 2001.

[120] Q. Wang, M. Wulfmeier, and B. Wagner. Voronoi-based heuristic for nonholonomic search-based path planning. In *The 13th International Conference on Intelligent Autonomous Systems (IAS-13)*, Padova, Italy, 2014.

[121] Q. Wang, C. S. Wieghardt, Y. Jiang, J. Gong, and B. Wagner. Generalized path corridor-based local path planning for nonholonomic mobile robot. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems (ITSC)*, Canary Islands, Spain, 2015.

[122] M. Overmars, I. Karamouzas, and R. Geraerts. Flexible path planning using corridor maps. In Dan Halperin and Kurt Mehlhorn, editors, *Algorithms - ESA 2008*, volume 5193 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin Heidelberg, 2008.

[123] M. Hentschel, O. Wulf, and B. Wagner. A GPS and laser-based localization for urban and non-urban outdoor environments. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, September 22-26, 2008, Acropolis Convention Center, Nice, France*, pages 149–154, 2008.

[124] D. Lecking, O. Wulf, and B. Wagner. Localization in a wide range of industrial environments using relative 3d ceiling features. In *Proceedings of 13th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2008, September 15-18, 2008, Hamburg, Germany*, pages 333–337, 2008.

[125] O. Wulf, D. Lecking, and B. Wagner. Robust self-localization in industrial environments based on 3d ceiling structures. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2006, October 9-15, 2006, Beijing, China*, 2006.

[126] E. Rohmer, S. P. N. Singh, and M. Freese. V-REP: a versatile and scalable robot simulation framework. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2013.

[127] S. Drogies and M. Bauer. Modeling road vehicle dynamics with modelica. In *Modelica workshop 2000*, Lund, Sweden, October 2000.

[128] P. Nobrant. Driveline modelling using mathmodelica. Master's thesis, Linkpings Institute of Technology, 3 2001.

[129] O. Amidi. Integrated mobile robot control. Technical report, Carnegie Mellon University Robotics Institute, 1990.

[130] J. M. Snider. Automatic steering methods for autonomous automobile path tracking. Technical report, Carnegie Mellon University Robotics Institute, February 2009.

[131] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney. Winning the darpa grand challenge. *Journal of Field Robotics*, 2006.

[132] A. de Luca, G. Oriolo, and C. Samson. *Robot Motion Planning and Control*. publisher, 1998.

[133] R. Murray and S. S. Sastry. Nonholonomic motion planning: Steering using sinusoids. *IEEE Transactions on Automatic Control*, 38:700–716, 1993.

[134] R. Murray. Control of nonholonomic systems using chained forms. *Fields Institute Communications*, pages 219–245, 1993.

[135] R. Lenain, B. Thuilot, C. Cariou, and P. Martinet. Adaptive and predictive non linear control for sliding vehicle guidance - application to trajectory tracking of farm vehicles relying on a single rtk gps. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pages 455–460, 2004.

[136] C. Cariou, R. Lenain, B. Thuilot, and M. Berducat. Automatic guidance of a four-wheel-steering mobile robot for accurate field operations. *Journal of Field Robotics*, 26(6-7):504–518, 2009.

[137] R. Lenain, B. Thuilot, O. Hach, and P. Martinet. High-speed mobile robot control in off-road conditions: A multi-model based adaptive approach. In *IEEE International Conference on Robotics and Automation*, pages 6143–6149, 2011.

[138] R. Lenain, B. Thuilot, C. Cariou, N. Bouton, and M. Berducat. Off-road mobile robots control: An accurate and stable adaptive approach. In *The 2nd International Conference on Communications Computing and Control Applications*, 2012.

[139] D. Dowson. *History of Tribology*. Professional Engineering Publishing, 2nd edition, 1997.

Publications

Q. Wang, M. Langerwisch, and B. Wagner: "Wide Range Global Path Planning for a Large Number of Networked Mobile Robots based on Generalized Voronoi Diagrams", *The 3rd IFAC Symposium on Telematics Applications (TA)*, November 11-13, 2013, Seoul, Korea. **Best Paper Award**.

Q. Wang, M. Wulfmeier, and B. Wagner: "Voronoi-Based Heuristic for Nonholonomic Search-Based Path Planning", *The 13th International Conference on Intelligent Autonomous Systems (IAS-13)*, July 16-18, 2014, Padova, Italy.

Q. Wang, C. S. Wieghardt, Y. Jiang, J. Gong, and B. Wagner: "Generalized Path Corridor-based Local Path Planning for Nonholonomic Mobile Robot", *2015 IEEE 18th International Conference on Intelligent Transportation Systems (ITSC)*, September 15-18, 2015, Canary Islands, Spain.

Curriculum Vitae

Name:	Qi Wang
Date of Birth:	20.01.1982
Place of Birth:	Shanxi, China

5/2014-Present:	Beijing Institute of Technology, School of Mechanical Engineering, Intelligent Vehicle Research Center, China.
9/2009-3/2014:	Leibniz Universitt Hannover, Institute for Systems Engineering, Real Time Systems Group, Germany.
9/2006-2/2009:	University of Science and Technology Beijing, Mechanical and Electronic Engineering, China.
9/2001-7/2005:	TaiYuan University of Science and Technology, Mechanical Design and Manufacturing Automation, China.