STUDIES IN AUTONOMOUS GROUND VEHICLE CONTROL SYSTEMS: STRUCTURE AND ALGORITHMS

DISSERTATION

Presented in Partial Fulfillment of the Requirements for

the Degree Doctor of Philosophy in the

Graduate School of The Ohio State University

By

Qi Chen, B.S.C.S.E, M.S.

* * * * *

The Ohio State University

2007

Dissertation Committee:

Ümit Özgüner, Adviser

Keith Redmill

Eylem Ekici

Approved by

Adviser

Graduate Program in Electrical and Computer Engineering

ABSTRACT

This dissertation studies issues in the implementation of the control system and path-planning algorithms in autonomous ground vehicles (AGV), which are designed for and further developed from both DARPA Grand Challenges in 2004 and 2005.

A hierarchical generic control architecture of the AGV control system is presented. In this architecture, the functionalities required for an AGV control system are decomposed into small and well defined tasks which are then carried out by different controllers. As an example, ION, the Intelligent Off-road Navigator, the autonomous vehicle participated in DARPA Grand Challenge 2005, is then introduced. Its control system structure and the hierarchical task decomposition are studied and designed. The discrete-event state machine in ION's navigation module and its physical layout of the vehicle hardware is also discussed.

In order to safely, smoothly and efficiently guide the vehicle to the goals, the navigation module in the control system is a very important part. The path-planning algorithms applied in ION's navigation module are then elaborated. We develop a real-time path-planning algorithm utilizing fuzzy logic. Two different fuzzy controller, fuzzy steering controller and fuzzy speed controller, are designed and discussed to ensure the planned path is goal-reaching and obstacle-free. In addition, because of the vehicle's physical constraints, we study the smooth path planning problem for the autonomous ground vehicles and the quartic spline interpolation method is

proposed. In order to make sure that the distance between the line segments of the given waypoints and the smooth path is small so that the vehicle stays right on the path, we formulate and study the minimum offset smooth interpolation problem. With the given way points and maximum turning curvature, the minimum offset can be calculated by using quartic splines. In the end we consider the maneuver planning problem. The vector space and state flow of the nonholonomic systems are defined. Three different approaches, Lie algebra approximation, dynamic programming, and nonlinear control method, are presented for different applications.

To Lu and Albert

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my adviser, Dr. Ümit Özüner, for his guidance and continuous support throughout my study. Especially, I want to thank him for providing me the precious opportunity to participate in the first and second DARPA Grand Challenge, as well as the advise and encouragement throughout these events. I would also like to thank Dr. Eylem Ekici and Dr. Keith Redmill for serving in my dissertation committee.

I want to thank Dr. Keith Redmill, John Martin, Dr. Charles Toth, Matt Knollman, George McWilliams, Rong Xu, Dr. Zhiyu Xiang, Yongjie Zhu, and everyone who worked and tested Terremax and ION in the Nevada desert. It is my pleasure to work with them. And I've learned a great deal from everyone.

I would like to acknowledge the financial support from the DARPA MICA program via AFRL, the Collaborative Center of Control Science and the Ohio State University.

Finally, I would like to thank all my friends and family. My special appreciation goes to my wife, Lu.

VITA

October 22, 1975	Born - Wenlin, Zhejiang, China
June 1997	B.S. in Control Science & Engineering College of Info. Science & Engineering Zhejiang University Hangzhou, China
January 2000	M.S. in Control Science & Engineering College of Info. Science & Engineering Zhejiang University Hangzhou, China
2000-2002	Project Manager/Senior Engineer Shanghai Research Institute of Huawei Technologies Shanghai, China
2002 - 2006	Graduate Research Associate Dept. of Elec. & Computer Eng. The Ohio State University Columbus, OH

PUBLICATIONS

Research Publications

Q. Chen and Ü. Özgüner, "Intelligent Off-road Navigation Algorithms and Strategies of Team Desert Buckeyes in the DARPA Grand Challenge '05," in *Journal of Field Robotics*, Volume 23, Issue 9, pp. 729-743, Sept. 2006.

Q. Chen and Ü. Özgüner, "Real-Time Navigation for Autonomous Vehicles: A Fuzzy Obstacle Avoidance and Goal Approach Algorithm," in *Proceedings of the 24th American Control Conference*, Portland, Oregon, June 2005.

Q. Chen and Ü. Özgüner and K. Redmill, "Ohio State University at the 2004 DARPA Grand Challenge: Developing a Completely Autonomous Vehicle," in *Intelligent Systems, IEEE*, Volume: 19, Issue: 5, pp. 8 - 11, Sept.-Oct. 2004.

Y. Hai and Q. Chen and Ü. Özgüner, "Control System Architecture for TerraMax - The off-road intelligent navigator," in *the 5th IFAC Symposium on Intelligent Autonomous Vehicles*, Lisbon Portugal, July 2004.

Q. Chen and Ü. Özgüner, "A Hybrid System Model and Overlapping Decomposition for Vehicle Flight Formation Control," in *Dynamics of Continuous, Discrete and Impulsive Systems, Series B: Applications and Algorithms*, Volume: 11, Number: 4-5, pp. 475–488, 2004.

Q. Chen and Ü. Özgüner,, "A Hybrid System Model and Overlapping Decomposition for Vehicle Flight Formation Control," in *Proceedings of the 42th IEEE Conference on Decision and Control*, Maui, Hawaii, pp. 516 - 521, Dec., 2003.

Q. Chen and X. Jin and S. Wang, "Application of Advanced Process Control in Aromatic Rectifying Tower," in *Information and Control* (Chinese Journal), Volume: 28, pp. 333-337, 1999.

Q. Chen and Y. Chen and S. Wang, "Computer Simulation and Application On Biochemical Process," in *Computer Simulation* (Chinese Journal), Volume: 16, No. 1, pp. 39-41, 1999.

Q. Chen and N. Wang and S. Wang, "An Improved Multi-layer ANN Learning Algorithm," in *Control Theory, Technology and Application* (Chinese Journal), Volume: 4, 1997.

FIELDS OF STUDY

Major Field: Electrical and Computer Engineering

TABLE OF CONTENTS

Page

LIST OF TABLES

Table **Page**

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

An Autonomous Ground Vehicle (AGV), also known as an intelligent vehicle, is a ground vehicle that can navigate in structured or unstructured environments without continuous human guidance. An AGV is able to detect and circumnavigate obstacles and finally reach a goal which can be a point, a region, or a sequence of points such as a trajectory.

The AGV's military development and application has already received a good deal of attention. In fact, the Department of Defense, including the Defense Advanced Research Projects Agency (DARPA), has supported robotics research at various levels of effort for almost two decades. Since the mid 1980s, DARPA has sponsored many autonomous land vehicle programs [1, 2], such as the DEMO I, II and III projects [3] and the DARPA Grand Challenges [4] including the DARPA Urban Challenge, scheduled to take place in November, 2007. Furthermore, in the 2001 Defense Authorization Act, the U.S. Congress set a goal that by 2015 one third of the operational ground vehicle would be unmanned [5]. Apart from their applications in military, AGVs also have variety of applications in civilian, such as wide-area environment monitoring, disaster recovering, search-and-rescue activities and planetary exploration, etc.

The research interest in AGV starts from the late 1960s. The first mobile robot *Shakey* with planning capability was built by Nilsson in 1969 [6]. The very first international joint conference on artificial intelligence was held in 1969. Since then, extensive efforts have been devoted to the development of autonomous robots. Furthermore, with the increase in the computation power and storage capacity of modern computers, highly complicated control system and precise models can now be implemented and work at high speed so as to make a powerful and intelligent vehicle a possibility.

This dissertation studies the issues in the implementation of autonomous ground vehicles control system. In particular, we focus on the control system structure design and the navigation algorithms. The control system structure (in Chapter 2, 3) and the real-time navigation algorithm (in Chapter 4) have been demonstrated by the Intelligent Off-road Navigator (ION), the intelligent vehicle developed by team Desert Buckeyes in OSU, which participated the second DARPA Grand Challenge held in the California Speedway and the Nevada Desert around Primm, NV in October 2005 [4, 7]. A variety of the design have also been demonstrated by TerreMax in the first DARPA Grand Challenge in April 2004 [8].

In this chapter, we will first overview the literature that is relevant to our study in control system architecture designs, path-planning algorithms and nonholonomic systems. Later an outline of this dissertation will be provided.

1.1 Literature in Autonomous Ground Vehicle Control Structure

Hierarchical structure is widely adopted in designing and controlling complex systems, such as in large-scale systems [9, 10, 11, 12] and robotic systems [13, 14, 15, 16]

including AGVs. In hierarchical structural designs, the system goal is decomposed vertically into a sequence of solvable tasks. There are several advantages in this hierarchical structure. First, the structure enables the designer to translate a big and complicated problem into several smaller and simpler subproblems. Furthermore, the error analysis and debugging process become easier and the cost for maintenance and improvement of the system is reduced since the modifications can be made at each local level instead of changing the whole system.

In the literatures, there are many robotic systems structured in hierarchy, because of the AGVs' high complexity and their dynamic environments, for example, the NIST 4-D/RCS reference model architecture [17, 18], the JAUS reference architecture [19], the ALV architecture [1] and the DAMN [15]. By doing so, the *behavior* types of abstractive functionalities are implemented in the high levels and, in the meantime, the low levels *react* to the environment changes fast enough so that the time delay is acceptable.

One type of the hierarchical control systems is called the *hybrid system* which includes the physical continuous system and the (software-driven) discrete system. Researches on hybrid systems have received a lot of attention since digital computers have been widely used in controlling complex systems, see [20, 21, 22] and reference therein. By implementing hybrid control systems, one can *design* the decision-makeing agents in the discrete states and *design* the continuous controller in the continuous states separately. In other words, the implementation of hybrid system is a *design* choice, because "it is convenient to exploit the degrees of freedom available ... it offers safety and performance guarantees *by design*, at least in an idealized situation." as stated by Frazzoli in [23]. In the autonomous vehicle's hybrid control

systems, the *behavior*s are defined in the discrete part as the higher levels in the hierarchy and the basic motor controllers are designed in the lower levels.

1.2 Literature in Path-Planning Problems

A basic problem, which many AGV control systems have to face, is the problem of *path-planning*, also known as *motion planning* problem. Path-planning is defined as a problem of finding a path from a start point to a goal point in a space while avoiding obstacles. At the late 1970s, the problem is first translated by Lozano-Pérez [24] into the problem of finding a connected path from two points in a space, called the configuration space (*C-space*). For example, any 3-D objects, such as a piano, are considered as a point in a six dimensional C-space [25]. For an AGV, which stays in a 2D-plane, its state can be represented by a triple $\xi = (x, y, \theta) \in \mathcal{C} = \mathbf{R}^2 \times S$, where $\mathcal{C}$ is the C-space, $S := [0,\ 2\pi)$, (x, y) are the Cartesian coordinates of a reference point, and θ is the heading angle with respect to the frame x-axis. The C-space is partitioned into C-obstacle and C-free subsets. A connected path connecting the origin and goal points is a solution to the path-planning problem if it is contained in the C-free subset.

According to Latombe [26, 27], the obstacle-free path-finding algorithms can be roughly divided into three groups: roadmap methods, cell decomposition methods, and the heuristic methods.

The road map methods construct network paths from the C-free subset, called "road map", so as to reduce the path-planning problem into finding (best) paths connecting the initial and final positions. A road map (RM) is a non-oriented graph obtained by topological retraction of the C-free subset. Path-planning is thus reduced

to connecting the initial and goal configurations by nodes in the RM and searching for the optimal path in RM according to certain criteria. Several methods based on the roadmap approach have been proposed, such as the *visibility graphs* [28, 29], *Voronoi diagrams* [30, 31, 32], *freeway net*, and *probabilistic planning* [33, 34] etc. The visibility graph method, explored by Nilsson [6] as early as 1969, is one of the earliest path-planning methods. It mainly applies to two-dimensional configuration space with polygonal C-obstacle regions. The Voronoi diagram method constructs the roadmap paths such that every point in the paths has equal distances to the nearest obstacles, and the roadmap paths always stay in the middle of the C-free and as away from C-obstacle as possible. Probabilistic planning is generally used when the configuration-space has more than five degrees of freedom, such as multi-link manipulators, mobile robots with trailers. There are two steps to construct the roadmap in the probabilistic planning approach: the first step is to efficiently sample the C-space at random and retain the free configurations at local minima; the second step is to check if each local minimum can be circumvented by a collision free path which forms a probabilistic road map. These roadmap-based approaches work well in well-known and static environments. When there is sensing noise, however, these approaches may generate totally different plans in two consecutive steps, because the vehicle dynamics are ignored.

The cell decomposition methods decompose the environment into obstacle-free cells and then find a sequence of neighboring cells that connects the initial and final condition [6, 35]. These methods assume the environment is known prior.

The potential field method is one of the most popular heuristic methods. In the potential field method, local movement of the robot is generated by "forces" that

expels the robot away from obstacles and attracts the robot to its goal [36, 37, 38]. The force field is bowl-shaped with the goal point at the bottom and has peaks around obstacles. By following the field gradient, the goal will be reached if not trapped in local minima. This technique offers speedup in performance but cannot offer any performance guarantee. The typical drawbacks in these algorithms are the existence of local minima where the robot may be stuck. To escape these local minima, techniques have been developed. For example, Barraquand and Latombe [39] use the randomized path-planner to escape local minima by executing random walks at the cost of computational efficiency.

There are other heuristic path-planning methods, such as genetic algorithm approaches [40] and fuzzy logic controls [41, 42, 43]. These methods cannot guarantee the *completeness* of the algorithms. We say a path-planning algorithms is *complete* if it can always find a path connecting the initial and goal, if such a path exists.

Stentz [44] proposes the D* (Dynamic A*) algorithm for performing dynamic path-planning in partially known environments. D* (Dynamic A*) performs the A* algorithm [45] initially on the entire workspace and dynamically updates certain components of the path when new information is obtained. By doing so, the shortest distance path to the goal is maintained according to the information available at any time instance. Because D* algorithm is based on a best-first graph search algorithm, it is complete. Nevertheless, the D* algorithm does not allow uncertainties in the sensed obstacles and position of the robot, which limits its applications in many situations.

There are also many rule-based behavior design approaches to solve the path-planning problem in the literature. Lumelsky et al. [46, 47] describe two algorithms

that can be used to solve the path-planning problem. Their algorithm, however, may require the robot to travel a long distance, equal to the Euclidean distance between two points plus 1.5 times of the sum of the perimeters of all the obstacles, which is not efficient at all. Daily et al. define the *Travel-Toward-Goal* and the *Find-Clearest-Path-To-Goal* behaviors in their path-planning algorithm, which is successfully implemented in [1]. Their system, however, limits the space of potential paths by only considering a static set of paths across the sensed area. This planner might find a region impassable simply because its static space of predetermined paths is insufficient to include a safe path.

The path-planning problem is of high complexity. Let alone that an AGV is nonholonomic and the environment may be only sensed locally, J. Reif shows the problem is PSPACE-hard [48], even in a well-known C-space for a holonomic system. The roadmap algorithm developed by Canny [49] solves any general path-planning problem with d degrees of freedom in $O(n^d \log n)$, which is the best general result for this free space approach, where n is the number of obstacle vertices, edges and faces.

Canny's algorithm generates near optimal path for polygonal body through a space containing polygonal obstacles. In the real application, however, the polygonal obstacle assumption sometimes is not satisfied. In fact, when we consider the application of an AGV navigating over rough natural terrains, no assumption about obstacle can be made. The obstacles may not even be discrete objects. Furthermore, the definition of obstacles may depend on the state of the vehicle. For instance, when an AGV faces a deep step, the upper plane is an obstacle if the AGV stays at the lower plane, and vice versa. As Brummit et al. address in [50], "when sensing must be done simultaneously with driving, the environment to the AGV is then uncertain."

Because prior knowledge of the environment may be incomplete or not exist at all and the environment may be dynamic, computing a path beforehand is impossible for many tasks.

Most of the above path-planning methods do not consider the mechanical constraints in the vehicles. Instead, the robots are considered to be able to move toward any direction at any time. In the late 1980s, Laumond et al. [51, 52, 53, 54, 55] point out that the geometric techniques developed in C-space cannot be applied directly in nonholonomic systems such as a car-like robot with minimum turning radius constraint. In other words, the existence of a connected path in C-space for a holonomic system is a necessary yet not sufficient condition for the existence of a feasible trajectory for a nonholonomic robot. Thus it brings more complexity to the problem of path-planning, since when a car-like vehicle is concerned, the nonholonomic property of the system has to be studied.

1.3 Literature in Nonholonomic System Path-Planning Problems

Nonholonomic systems are a special case of under-actuated systems whose constraint equation is not completely integrable. In other words, for a driftless control affine system, even if the number of controls is smaller than the dimension of state space, it is still possible to reach all of the state space [56, 57]. For example, a car-like robot modelled by equation (1.1) is a nonholonomic system. With two controls, the linear velocities v and steering angular ϕ [1], it is able to move to any point in the

[1]The steering angle is defined by the angle between the average of front wheels and the main direction of the vehicle. Practically, we consider that $|\phi| \leq \phi_{max} < \pi/2$.

C-space of 3 dimension, $\mathbf{R}^2 \times S$.

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \cos\theta \\ \sin\theta \\ 0 \end{pmatrix} v + \begin{pmatrix} 0 \\ 0 \\ \frac{1}{L} \end{pmatrix} v \tan\phi \tag{1.1}$$

The vehicle is constrained by (1.2) since the tires are not allowed to slide, in which case, the vehicle can only move toward its tangent direction of its trajectory. As a consequence, a path feasible for holonomic system, such as an omni-directional robot, may not necessarily be a feasible path for the system.

$$\dot{x}\sin\theta - \dot{y}\cos\theta = 0 \tag{1.2}$$

As a pioneer work, in 1957, Dubins obtained the shortest path in connecting two configurations with bounded curvature [58] while regarding the car-like robot moving only forward. Later in 1990, Reeds and Shepp tackled the similar problem with a car-like robot that can move both forward and backward. Both Dubins and Reeds-and-Shepp's shortest paths consist of straight line and arcs with minimum radius. These results generate a minimum distance trajectory that connects two configurations and satisfies the constraint of (1.2). Nevertheless, the curvature along the trajectory is not continuous.

Many efforts on nonholonomic path-planning have also been paid on *smoothing* the path [59, 60, 61]. A smooth trajectory is a trajectory along which the derivative of the tangent vector is piece-wise continuous. In other words, the trajectory is derivative and its curvature is continuous. This property is also known as the *second order geometric continuity* [62]. When a trajectory is not derivative, it is impossible for a car-like robot to follow since it cannot change its direction without changing its position. Moreover, if the curvature of the trajectory is discontinuous at a (tangent) point, then the robot has to stop and reorient its front wheel steering angle.

In [61], the authors apply the cubic Bezier curve, to smooth the trajectory. But there is no suggestion on the selection of the control points and it does not work in the situation where the lateral distances to obstacle are short. In [63], the authors solve the optimal problem according to the performance evaluation, the curvature and jerkiness. The cubic spirals are obtained to connect two configurations. They claim that their vehicle moves far smoother than when clothoid pairs are used as in [64]. The authors in [62, 65, 66] propose to use a quintic spline to interpolate two points with given direction and curvature at each points. Since quintic spline is second order geometric continuous, the authors call the result path "G^2-splines". The use of the quintic G^2-splines provides several degrees of freedom in selecting parameters to improve the quality of the trajectory. The authors, however, do not discuss how to choose the direction and curvature at each connecting point. Instead, these conditions are provided as the problem setup. In addition, none of the above smoothing approaches have considered the displacement of the trajectories from their original non-smooth paths. If the displacement is large, an obstacle-free path may encounter obstacles after smoothing.

1.4 The DARPA Grand Challenge Overview

The DARPA Grand Challenge (DGC) is a prized competition for AGVs, sponsored by the Defense Advanced Research Projects Agency (DARPA) [4]. In the first two DGCs, thousands of waypoints (GPS coordinates) are provided shortly before the race, and the AGVs are required to follow the waypoints one by one safely, continuously, smoothly, and fast across natural terrain en route to the finish line without any human interference. The challenge is indeed "grand" because the AGVs have

to respond to the dynamically changing environment in a timely way and the data acquired is complex and contains uncertain information [7]. The competitors are intended to accelerate research and development activities in autonomous ground vehicles, increase collaboration among different organizations and interest groups, and draw widespread attention to the technological issues of developing completely autonomous ground vehicles.

In March 2004, DARPA held the first Grand Challenge (DGC04), a 142-mile mostly off-road race for 19 unmanned vehicles through an unrehearsed route across the Mojave Desert from Barstow, CA to Primm, NV. At the Ohio State University, we formed the Team TerraMax in collaboration with Oshkosh Truck Corporation, who provided an MTVR (Medium Tactical Vehicle Replacement), an 8×28 ft., 32,000 lb. truck manufactured for off-road military applications. TerraMax was installed with drive-by-wire technology and modified to fulfill DARPA safety requirement. Sensor fusion systems, high-level navigation system, low-level vehicle controller, visual road detection system, map-based path-planning algorithm, and real-time communication among computers were developed for the vehicle. As a result, TerraMax was one of the only seven vehicles that completely traversed the Qualification, Inspection, and Demonstration (QID)[2] course. In the main challenge event, TerraMax traveled 1.2 miles within the required corridors.

In October 2005, the second Grand Challenge (DGC05) was held in the Mojave Desert along a route of 132 miles near Primm, NV. As Team Desert Buckeyes from the Ohio State University, we developed an intelligent off-road autonomous vehicle,

[2]The QID in the 2004 DGC was the 1.5 mile test track DARPA used as the semifinals. The test track consisted of most of the terrain types and obstacles that an AGV would likely meet along the DGC course, including dirty hills, ditches, sand pits, tunnels and underpasses, cattle guards, boulders, towers, and even moving obstacles.

the Intelligent Off-Road Navigator (ION). ION is a six-wheeled vehicle with full drive-by-wire capability. A set of external environment sensors (LIDARs, radars, cameras and ultrasonic transducers) and vehicle ego-state sensors (a GPS receiver, a vertical gyroscope, a digital compass, and wheel speed sensors,) provided extensive sensing capability for the vehicle. A sophisticated sensor fusion system [67] has been developed and used with a complex intelligent analysis, decision and control configuration [68]. ION was developed in partnership with University of Karlsruhe, who provided the image processing system [69]. A number of aspects of ION were also descendants of technology and approaches used in the 2004 Grand Challenge. As one of 43 AGVs to participate the National Qualification Event (NQE) before the DGC05 race, ION completely traversed the NQE[3] course successfully four times[4], which fully exhibited ION's obstacle-avoidance and goal-approaching capabilities. In the DGC05, ION covered about 30 miles in the Nevada Desert with complete autonomous operations, and finished as the 10th furthest.

The third Grand Challenge, which is also known as the DARPA Urban Challenge, will take place in November 2007. In contrast to the previous two DGCs, the Urban Challenge requires AGVs to be able to navigate in a road network of an urban area and obey all traffic laws. AGVs are also expected to encounter other vehicles and objects on the course. This challenge is "grand" too, because AGVs must analyze the situation and make "intelligent" decisions in real time, based on the actions of other vehicles.

[3]The NQE in the 2005 DGC was the 2.7 mile test track DARPA used as the semifinals. The track provided a series of obstacles and the AGVs are required to go through 51 "gates". It also had a 100ft metal "tunnel" where GPS was lost.

[4]ION tried the NQE course five times in total and only failed once.

The autonomous vehicles, TerraMax and ION, are the result of the efforts from many individuals. I am the designer of ION's software structure and high level controller. In fact, the major part of the this dissertation is developed and concluded from the implementation of control system structure and the navigation module in ION.

1.5 Dissertation Outline

In this dissertation, we discuss issues in the implementation of the control system of AGVs, with particular focus on the system structure and path-planning algorithms in the navigation module.

The dissertation is organized as follows. We begin with the AGV control system structure. Chapter 2 describes a generic hierarchical architectures that is suitable for the implementation of a class of AGVs. The system structure, the software structure, the navigation module structure of ION are presented in Chapter 3. Chapter 4 describes the real-time local path-planning algorithm and its implementation in ION for generating obstacle-free and goal-reaching local path. Chapter 5 studies the problem of smoothing so that the planned trajectory for vehicle to follow is of second order geometric continuity. Chapter 6 presents an automatic maneuver planning algorithm which generates a sequence of "unit maneuver operations" to steer the AGV from any state to any desired configuration amidst obstacles. Chapter 7 concludes the dissertation and provides a summary of the current work and discussions for the future researches.

CHAPTER 2

GENERIC ARCHITECTURE FOR AUTONOMOUS GROUND VEHICLE CONTROL SYSTEMS

2.1 Introduction

This chapter discusses the autonomous ground vehicle control system from an architectural perspective.

Since autonomous ground vehicles are developed for different purpose, the AGV system structures vary from one to another. Most of them are closely related to the tasks they are designed to perform. For example, AGVs that are developed for off-road missions may not be able to navigate in urban areas, obey traffic rules and co-exist with other vehicles; AGVs that are developed for structured environments probably cannot survive in off-road scenarios; Some AGVs may work well independently yet fail to cooperate with other AGVs.

On the other hand, although AGVs are heterogeneous, they share common components in their control system. For example, as an intelligent system, the AGV has to be able to sense its own status, detect obstacles, and implement proper control actions through actuators.

In this chapter, we are going to introduce a generic structure for AGV control systems. Four levels with different requirements and objectives are presented.

2.2 General Requirements for AGV Control System

In general, an AGV control system's task is to guide and control the vehicle to reach the goal successfully in an uncertain and dynamic environment. The AGV control system is able to sense the environment locally and perceive the situation in order to make proper decisions. A working AGV control system should be able to:

- accept external commands, such as *pause*, *stop* and *start/restart*, and respond properly;
- recognize mission descriptions in defined protocol;
- stabilize the vehicle: control the vehicle to follow a line at specified speed within acceptable error range;
- sense AGV's own status and environment through various sensors and sensory processing algorithms;
- generate feasible obstacle-free paths toward goal(s) in real time;

and some may even be able to:

- perceive the current situation in order to make proper decisions;
- learn from driving data and adapt to the environment;
- comply with traffic rules or other predefined rules;
- detect and react properly to abnormal events or situations;

- perform the task optimally according to predefined criteria such as minimum time, minimum fuel, maximum safety, etc.;
- cooperate with other agents through communication, where other agents can be other AGVs, human-driven vehicle, console computers;
- allow mission changes from commander via communication;

To fulfill the above requirements in AGV control systems, we need to deal with high complexity and rich details in short time and with limited computing resources. Because the hierarchical structures have advantages in dealing with highly complex systems, it is not surprising to find out that most of the AGV control systems are structured in hierarchy, such as the NIST-RCS reference model architecture [17, 18], the ALV architecture [1], the DAMN [15], and AGVs participated the DGCs [16], etc.

2.3 The Hierarchical Generic Architecture

In [9, 10, 11, 12, 70], the authors develop a framework for hierarchical structures: by physically or functionally decomposing the system and by associating functionality to the structure. In the hierarchical goal-task decomposition, the higher level's *Goal* is decomposed into several *Tasks*, each of which is easier to accomplish than the original *Goal*, and these *Tasks* are then assigned to the lower level as its *Goal* to accomplish. By achieving the *Goals* in the lower level, the *Goals* in higher level are achieved. Then in the end, the entire system *Goal* is accomplished.

Figure 2.1 shows the *Generic Architecture* that is suitable for implementation of a class of AGV control systems. The architecture is a hierarchy that can be roughly divided into four levels, *strategic level*, *tactical level*, *skill level* and *operational level*, as

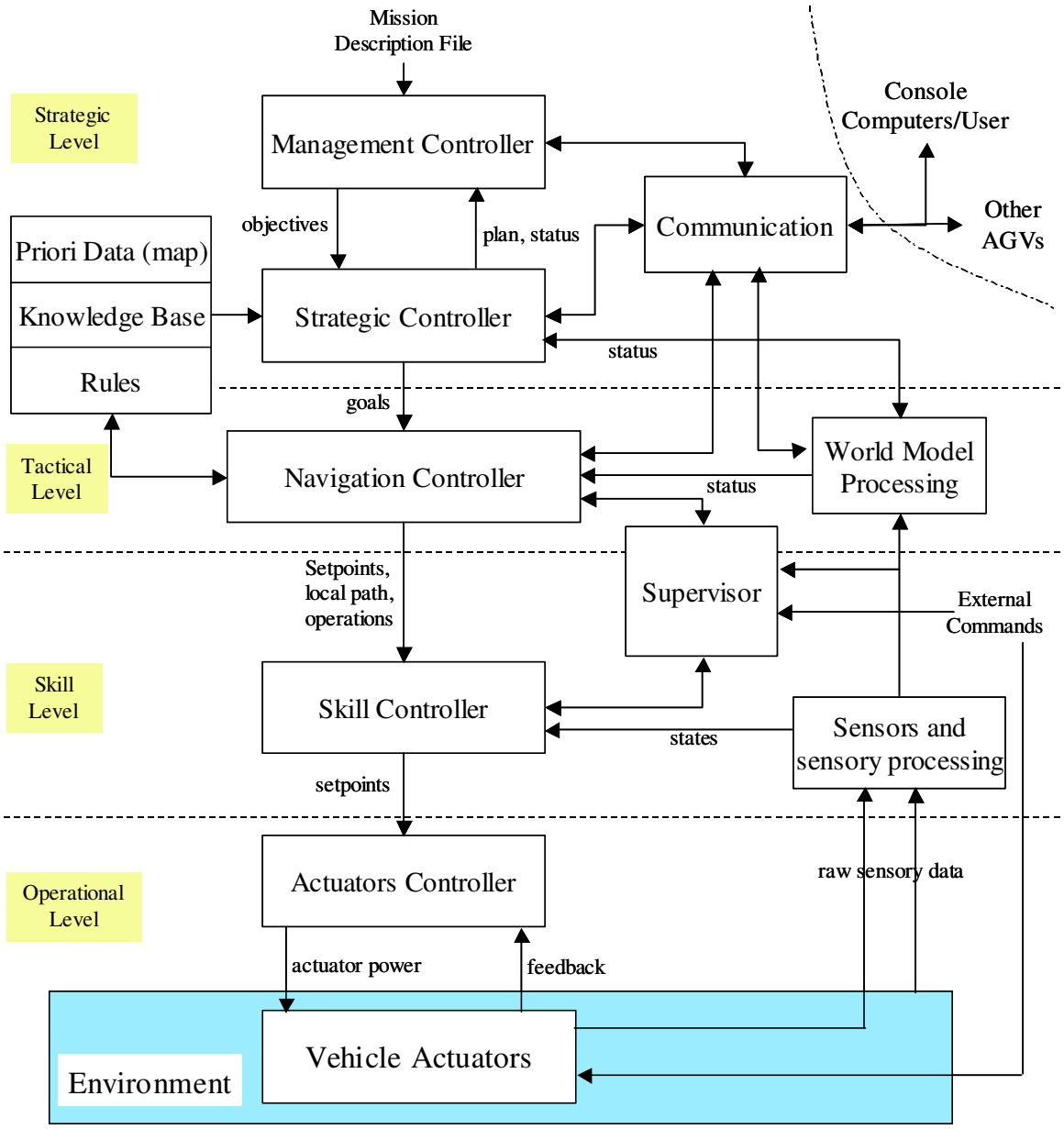


Figure 2.1: The generic architecture for a class of AGV control systems.

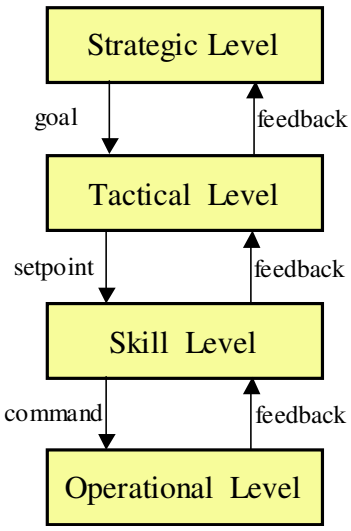


Figure 2.2: Abstract hierarchy structure of the generic architecture.

shown in Figure 2.2. The higher levels in this hierarchical structure tend to require longer cycle times and be more logical, abstract, and goal-oriented than the lower levels. In contrast, the lower levels are more reactive and run at higher frequency than the higher levels.

2.4 Layer Descriptions

2.4.1 Operational Level

The *operational level* is the fundamental level for controlling an AGV. In this level, drive-by-wire technique is installed so that the *wheel-turn angle, throttle position*, and *brake force*[5], are controlled and stabilized to desired setpoints through *actuator controllers*. The *gear* can shift to the desired position too. In addition, in some AGVs, external commands that relate to safety issues are taken care of in this level.

[5]For electric-powered vehicles that are driven by electric motors, their speed is determined by the motor supply voltage, or PWM ratio, which is equivalent to the throttle position in engine-powered vehicles. This difference does not affect the generic structure discussed in this chapter.

The *operational level* provides the basic functionalities for controlling the vehicle. It concerns controlling the actuator to do exactly what it is told to do, by dealing with the (nonlinear) actuator dynamics in the presence of disturbance. The dynamics of actuator is nonlinear not only because the physical hardware is nonlinear, for example, the nonlinearity in DC motors and gears, but also because the dynamics of the system depends on the vehicle states such as the speed and the gear position.

As the lowest level in the control system hierarchy, the *operational level* is expected to respond to the reference change stably and with very short response time. Utilizing the sensing signals directly without heavy processing makes it possible for the actuators to follow the references quickly and faithfully.

In contrast to the higher levels, it is not able to make "intelligent" decisions, such as applying maximum *brake force* to avoid collisions.

2.4.2 Skill Level

The *skill level* consists of the *skill controller* module and the *sensor and sensory processing* module. This level's goal is to control the vehicle to follow a local trajectory at a given speed. Both the local trajectory and the velocity setpoint are generated by the *navigation* module at the higher level, the *tactical level*, as discussed in the next section.

Skill Controller

In most implementations, the *skill controller* consists of a *longitudinal controller* and a *lateral controller*. The former is responsible for maintaining the vehicle speed at the given setpoint by generating *throttle position* and/or *brake force* setpoints, as well as the *gear position*. The latter generates the setpoint of wheel-turn angle so

that the vehicle can follow the given local trajectory. These setpoints are then sent to the *operational level* to be followed by the actuator controllers. We can also develop *direct controller* in this module that carries out certain commands from higher level with the specification of a sequence of preplanned motions that we called *robotic unit operations*, such as "maximum-left-turn-half-throttle-5-seconds".

The development of these controllers utilized the technologies of automatic control theory. One of the *longitudinal controller*, also known as *cruise control*, has been extensively studied, and commercial vehicle equipped with cruise control started as early as 1958. Considerable research and development efforts have been devoted to developing *lateral controllers* and fruitful results have been obtained, see [71, 70, 72, 73, 74].

Sensor and Sensory Processing

The other important component in this level is the *sensor and sensory processing* module. This module collects data from various types of localization and orientation sensors, including GPS, gyroscope, compass, wheel-speed sensors, and other internal sensors. Extended Kalman Filters (EKF) [75, 67, 76] or other filtering and state estimation algorithms are used in this module to generate vehicle ego-states, such as 3D-position, speed, acceleration, yaw, roll, pitch, and their rate, etc. The estimated vehicle ego-states are then passed to the *longitudinal controller* and the *lateral controller* to form a feedback loop. The external environment information, together with vehicle's ego-states, is sent to the upper level, where sophisticated sensor fusion system is implemented to build the *world model* which will be described in the next section.

supervisor module may be introduced into the *skill level*, as presented in Figure 2.1. This module monitors the vehicle's ego-states, in-coming external signals and certain events, such as degraded GPS quality or GPS blackout, sensor failure, etc. By utilizing the *supervisor* module, the *skill level* is allowed to have limited authority to make "intelligent" decisions, such as reducing speed setpoint because of fast yaw rate or large roll angle to keep the vehicle from rollover.

We call an AGV's control system to have a *minimum setup* if it only contains the *operational level* and the *skill level*. An AGV (or a ground robot) with a *minimum setup* possesses a certain degree of automation and maybe used in simple and static environments, for example, the obstacles are fixed and an obstacle-free path is predefined. However, it cannot *react* to the environment around it or deal with abstract tasks such as "go to point A" without specifying the detailed path because of its simple setup.

At this level of the control system hierarchy, both the controllers and the sensory processing run at a high frequency so that the response time is small, which is necessary for stably controlling fast moving vehicles.

2.4.3 Tactical Level

There are two major components in the *tactical level*: the *world model processing module* and the *navigation controller*.

World Model Processing Module

The *world model processing* module fuses various sensory data from the vehicle ego-state sensor fusion and from multiple external environment sensor signal, including laser range-finder (LIDAR), RADAR, camera and image processing system, and

ultrasonic rang-finder (SONAR), etc. Feature information, such as obstacles, terrain type, road edges, is then extracted into a map representation, called *world model*. Extra information, such as prior known road networks, GPS maps, and aerial maps, can also be incorporated into the *world model* in this module. As the system's internal representation of the external world, the *world model* acts as a bridge between the sensory processing and the rest of the system by providing a unified central representation of the sensory data. Sophisticated sensor fusion systems are implemented to generate the world model map and keep it updated and consistent. For example, in ION, a cell-based map was generated by using multiple confidence measures and estimated object height. The world model processing module was developed to scroll the map as the vehicle moves, to update map data, to fuse data from redundant sensors.

In addition, we may also want the world model processing module to be able to predict the future consequences of the vehicle states under certain navigation planning.

Navigation Controller

The *navigation controller*, also called the *navigator*, uses the world model map to plan a safe, efficient and feasible local path from the current position to the commanded goals in real time. By *feasible*, we mean the local path should satisfy the constraints imposed on the AGV system, and should be able to be closely followed by the *skill level* controllers, for example, the local path should be smooth. By *efficient*, we mean the local path should be (locally) optimal according to the predefine criteria so that the goal is reached efficiently. And most importantly, the local path should be *safe* so that by closely following the path, the vehicle is collision-free. The local

path-planning algorithm implemented in the *navigator* should be able to maximize the probability to find such a local path, if it exists. When such a local path cannot be found, a sequence of *robotic unit operations* will be generated to maneuver the vehicles from one configuration (position and heading) to another configuration where a safe and feasible local path leading to the goal can be generated to start from.

In addition to the world model map, navigators are allowed to access extra information such as priori map data, predefined schema or recipes stored in knowledge base. Predefined rules may be applied to the navigator. For example, AGVs may be designed to follow traffic laws, such as to bear right on two-lane road, to yield main traffic flow while merging into, etc. The vehicle status and external commands collected in the *supervisor* module may also be used in the navigation module.

One concern of the *navigator* is the computation time and resources required to generate a local path. On one hand, the *tactical level* tends to deal with complicated situations and with detailed information about the environment, so that it usually requires more time and more computing resource than the *skill level* and *operational level*. On the other hand, the navigator also needs to *react* to the environment changes by replanning the local paths timely in order to avoid colliding into (lately) detected obstacles, so that the *tactical level* cannot afford too much time for generating a local path, especially when fast moving AGVs and dynamic environments are considered. Thus, sophisticate navigators that require extreme computation are not suitable for systems with limited computation resources. When designing a navigator, trade-offs are often made between rich information utilization and quick reaction capability.

The other concern of the path-planning algorithm is the ability to operate on poor quality data. The world model cannot represent the real world accurately and

precisely, due to various of types of uncertainties and noises, such as the physical noise from the sensor, the asynchronism in different sensors, especially when vehicles run fast on bumpy terrains, or some unmeasurable "parameters" of the world. As a result, noisy information, such as false obstacles and undetected obstacles, are inevitable in the world model representations, though they can be maintained at a low level. In this case, the path-planning algorithm has to be able to operate along with the uncertainties as reliable as possible.

In contrast to the lower levels in the control hierarchy, the *tactical level* is able to make intelligent decisions concerning the vehicle-environment relationship. Fast inter-AGV cooperation capabilities can be considered in this level. For example, when two or more AGVs are close to each other, the navigators would consider the other navigators' planned path(s) while generating their own local path to avoid collision. These fast inter-AGV cooperation might be established by communicating with each other, or by predefined protocols like traffic laws. The advanced intelligences, such as global path planning, mission-based inter-AGV cooperation, pursue-evade game, and mission analysis, are considered in the *strategic level*.

2.4.4 Strategic Level

The *strategic level* concerns mainly on problem-solving issues by setting a sequence of goals, while the lower levels concern on controlling the vehicle to achieve these goals. This level is virtually omnipotent in this generic architecture. In real implementations, however, the capabilities that this level is designed to have are very much task-based or scenario-based.

The *strategic level* itself is also hierarchically structured. The *management controller* is used to decompose the mission which is either obtained from digital mission description file or from console computer/human through communication module, into a sequence of objectives. These objectives, if more than one, are then scheduled and passed to the *strategic controller* one by one. These objectives can be highly abstract, for example, "search for target **X** in area **D**" or "go to position **P**", etc.

The *strategic controller* generates tangible plans for the objectives, for example, a sequence of waypoints connecting place A to place B is a tangible plan for getting to B from A, where each waypoints in the plan becomes a goal for the *tactical level* to reach.

In addition, the plan is usually (globally) optimal according to the predefined criteria and currently available data. When the world model and vehicle status changes significantly so that the current plan is no longer feasible, the objectives and the plans should be changed. Furthermore, they can also be overridden by commands from console computer/user.

In order for human user to monitor the process, it is common to have mission objectives, strategic plans and latest vehicle status represented via user interface. It is also common to broadcast the AGV status and its plan in a cooperative scenario. These kinds of information exchanging can be implemented in this level.

At the top of the control system hierarchy, the *strategic level* runs at low update rate. Therefore, it does not respond quickly to the world model changes. Moreover, in some simple applications, controllers in this level may run only once at the beginning of a mission. Or the planning process can even be done off-line before carrying out the mission.

2.5 Cooperations Among AGVs in the Generic Architecture

In addition to being applied in a single AGV case, the generic hierarchical architecture framework presented in this dissertation can also work in the cooperation scenarios. In this subsection, we will discuss the hierarchical architecture framework for cooperative AGVs. Three types of cooperative interactions, which are included in many general multi-AGV cases, are introduced. In order to be more general, in this subsection we will use agents to represent AGVs. Of course, AGVs are one kind of agents. Agents, however, can be used in more general intelligent entities. Depending on their physical setup or applications, agents may cooperate under one type of interactions. In more complicated cooperation cases, two or three types of interactions may exist at the same time. Figure 2.3 shows how agents interact with other agents via these three types of interactions.

The first type of interaction, shown in Figure 2.3 as Type 1, is the relation between the commander agents and other agents. The commander agents are those sending out commands for others to follow. Human operator can also control agents through a commander agent. Either a commander agent or the one receiving human commands controls the other agents through the strategic level. It modifies or overrides the plans that the tactical level follows. By doing so, the human operator or the command agent can master other agents to achieve his/her/its objectives. Usually, this type of interaction is considered to be a centralized cooperation setup, since the commander agent sends plans to other agents. And the other agents may not generate their own plans, or their plans are overridden by the receiving assigned ones.

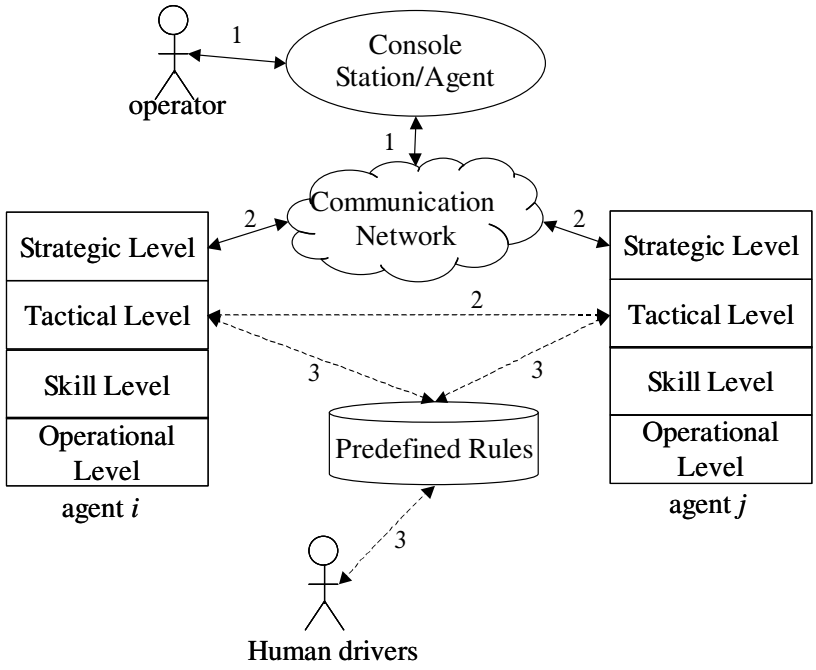


Figure 2.3: The cooperative interaction among agents with hierarchical structure and with other agents. 1) Human operator controls the mobile agents through a centralized station agent; or, the station agent dispatches plans to the strategic level of each agents, in a centralized cooperation way; 2) Agents exchanges states and plans in the strategic and tactical level, and negotiates in a decentralized cooperation way; 3) Agents and human drivers "cooperate" under predefined rules.

The second type of interaction, shown in Figure 2.3 as Type 2, is the one among agents that are usually seen in a decentralized cooperation setup. Each agent generates its own plan in the strategic level. The agents may interact with each other in the strategic level, in which case they exchange their own states and plans and maintain other agents' states and plans in the world model representation as well as their own ones. After negotiating with each other through the communication network, the controller in the strategic level modifies the plans for the benefit of its goal. When agents come close to each other, where fast response is needed, a direct

communication link in the tactical level is built. By exchanging states and plans in this level, the agents can respond more quickly, which increases their safety, since the tactical level runs faster than the strategic level.

The agents may also interact with each other without any direct communications, which is the third type of interactions we consider. The predefined rules can be adopted in the tactical level so that the agents cooperate naturally under these rules. For example, each agent should stop at "stop" signs and yield to previously stopped vehicles, etc. In this case, by adding the traffic rules of human drivers, the agents can cooperate or co-exist with each other and more importantly with manually driven vehicles.

Basically, in our cooperation hierarchical structure, the strategic level is for cooperative planning, while the tactical level is used for cooperative execution.

CHAPTER 3

ION'S CONTROL SYSTEM STRUCTURE

The hierarchical structure discussed in the previous chapter provides us a guidance to develop an AGV. In this chapter we will introduce the control system structure of an intelligent vehicle, ION, that is designed based on the generic hierarchical structure. ION, the Intelligent Off-road Navigator, is the vehicle demonstrated in DARPA Grand Challenge 2005. ION's physical layout and its functional architecture will be presented in this chapter as well.

3.1 The Physical Layout and Functional Architecture of Vehicle Hardware

The physical layout of ION is shown in Figure 3.1. ION is a six-wheeled vehicle with full drive-by-wire (DBW) capability. A set of external environment sensors, including one vertical laser range-finder (LIDAR), three horizontal LIDARs, one Radar, two cameras and eight ultrasonic range-finders (Sonars), are mounted on the vehicle as indicated in the figure. The vehicle ego-state sensors include a precision GPS receiver, a vertical gyroscope, a digital compass, and wheel speed sensors. The gyroscope and three high performance computers are mounted inside the computer cabinet. Digital compass is attached to the cabinet surface. DBW electronic control

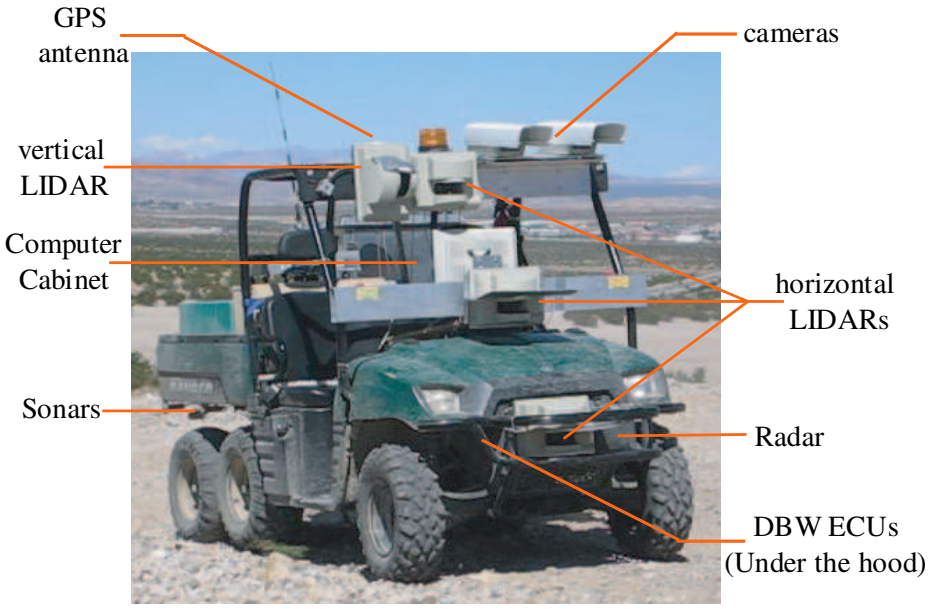


Figure 3.1: Physical layout of ION's hardware. The vehicle is a 2005 Polaris Ranger 6x6. It is 120 inches long and 60 inches wide and its height is 78 inches. Drive by wire capability has been added to the vehicle so that computer control is possible for throttle, brake, steering control, and transmission gear switching.

units (ECUs), a micro controller and actuators are mounted under the hood. Detailed description about the sensing system components, as well as the sophisticated sensor fusion system developed for ION, can be found in [67, 68].

Figure 3.2 shows the functional architecture along with the hardware layout. The deployment of these functional modules concerns the balance of the computation burden in each computer. The high level controller (HLC), the low level controller (LLC) and the ego-state sensing and fusion procedures are allocated in the central computer whose operating system is QNX. The stereovision system is allocated in the image-processing computer that runs on Windows XP, which provides the obstacle information detected from vision data to the world model processing. The ego-state

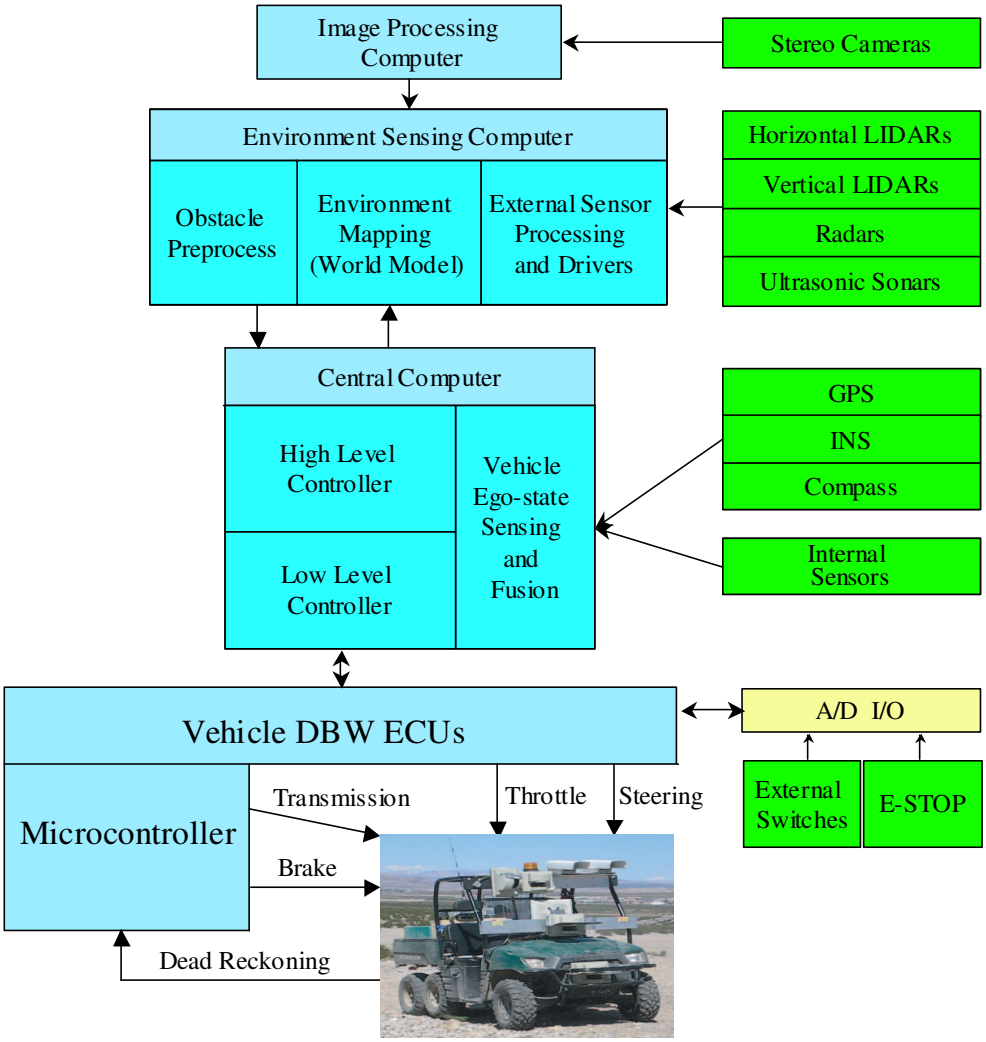


Figure 3.2: ION's functional architecture along with the hardware layout.

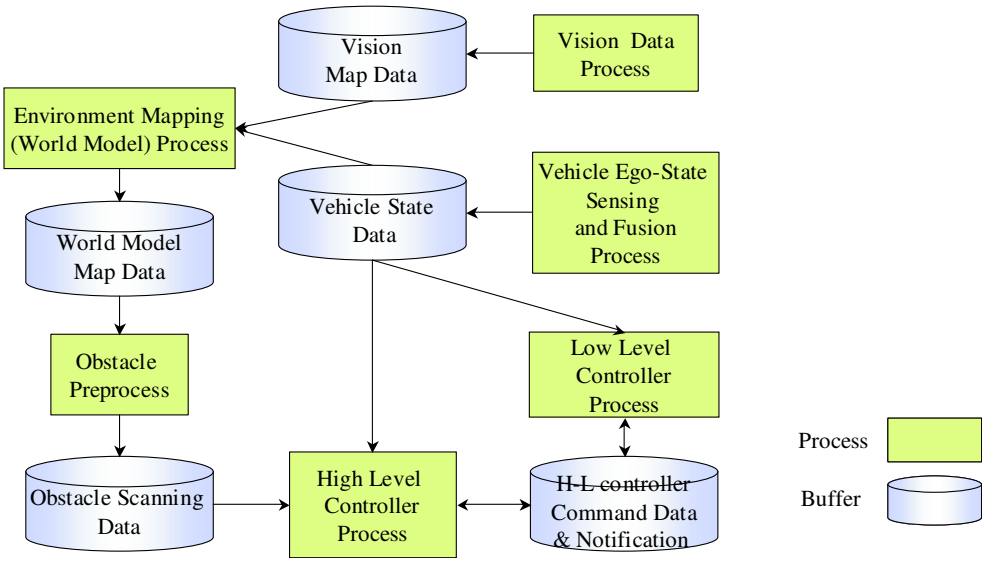


Figure 3.3: Data communication between the autonomous-driving-system processes.

sensing and fusion result is sent to the world model processing too. By incorporating various kinds of data, a world model map is generated to represent the environment around the vehicle. The sensor fusion system runs in the environment-sensing computer whose operating system is LINUX. Shared memories are used for data exchanging among processes in the same computer. User Datagram Protocol (UDP) is used for inter-computer data exchanging. To reduce the burden on the local network for transferring huge world map data at a high frequency[6], the obstacle information in the map is preprocessed before being sent to the central computer. Figure 3.3 shows the relationship between the processes and the shared memory buffers.

[6]A cell-based map is used in ION as the world model map. Each point in the map, an eight-bit data structure, represents a 0.2×0.2-meter cell so that the 80×40-meter local map that ION uses results in a 400×200 byte data array. The map needs to be updated and transferred at 10Hz.

3.2 Hierarchical Task Decomposition

Figure 3.4 shows the hierarchical *Goal-Task* decomposition in ION's control system, which is constructed based on the generic hierarchical architecture framework. In the *strategic level*, there is the "race planner", corresponding to the "management controller" in the generic model. It loads the road-definition-file (RDDF) and provides the goal and parameters of each section to the "section planner" at the beginning of each races. The "section planner" is the "strategic controller" in the generic architecture. It monitors the vehicle situation and provides goals to the "navigation controller" when the previous goal is reached. In the *tactical level*, the "navigation controller" module is further decomposed into "mode planner", "path planner", "robotic unit operation planner" and "rollback planner". The *skill level* controllers include the "lateral controller", "longitudinal controller" and "robotic unit operator". These modules run at different time scales, as indicated in the left column of Figure 3.4.

3.3 The Discrete-Event State Machine in the Navigation Module

As discussed in the generic architecture in Chapter 2, the navigator in the *tactical level* is responsible for analyzing the situations, planning feasible, efficient and safe local paths, and dealing with discrete events. In addition, the navigator has to run at the short time scale so that it can respond promptly to the environment changes.

To deal with all situations that ION may encounter, we develop ION's navigator in two levels. The upper level is a finite state machine (FSM) driven by discrete events, while the bottom level consists of several subsystems, each of which can be

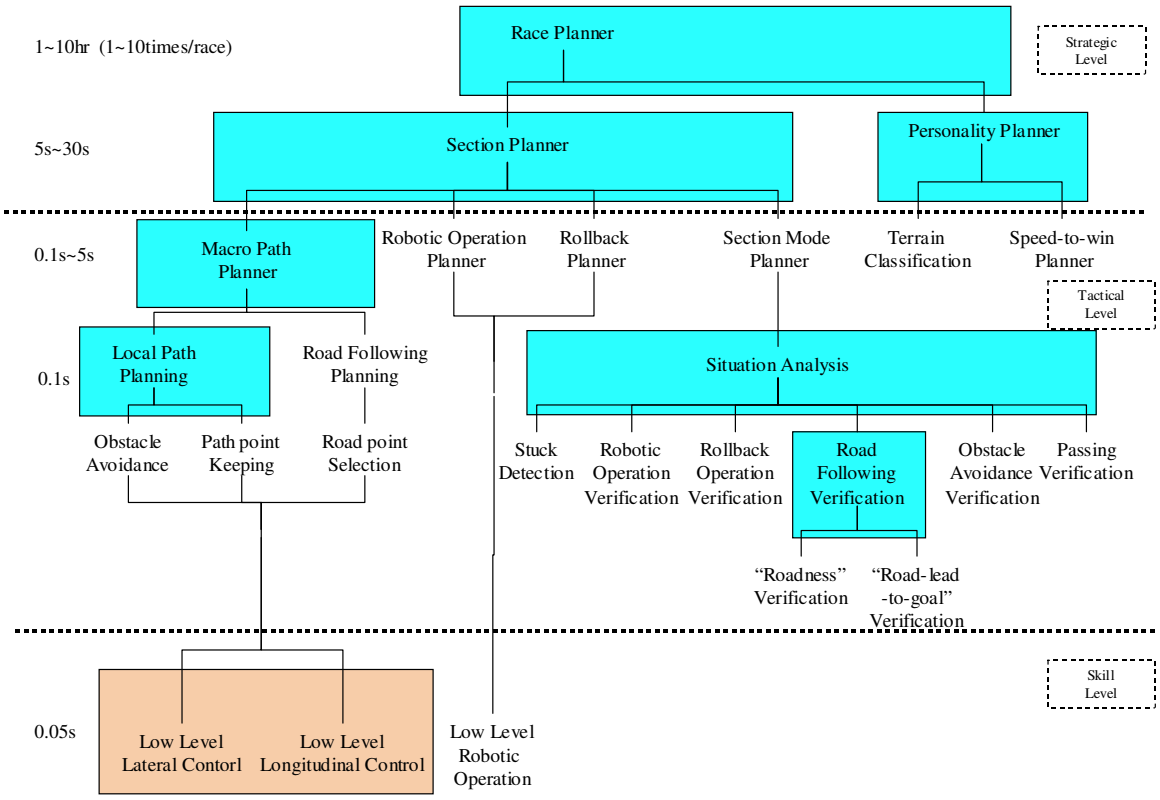


Figure 3.4: The functionally hierarchical decomposition of ION's control system. The functional modules run at different time scale, indicated in the left column.

used to generate path planning or operation planning commands. Based on the events obtained from the "world model" and the "supervisor" module, the navigator switches to the right machine state and activates the corresponding subsystem, so that the proper setpoints are generated for the LLC.

The FSM is represented conceptually in Figure 3.5. In the central meta-state, there are three major states, the *path-point keeping* state, the *obstacle avoidance* state and the *road following* state. ION stays in this meta-state at the most of racing time, when no special events, such as tunnel, sensor failure, and narrow path, etc., are detected. When the FSM state stays in the central meta-state, the path-points

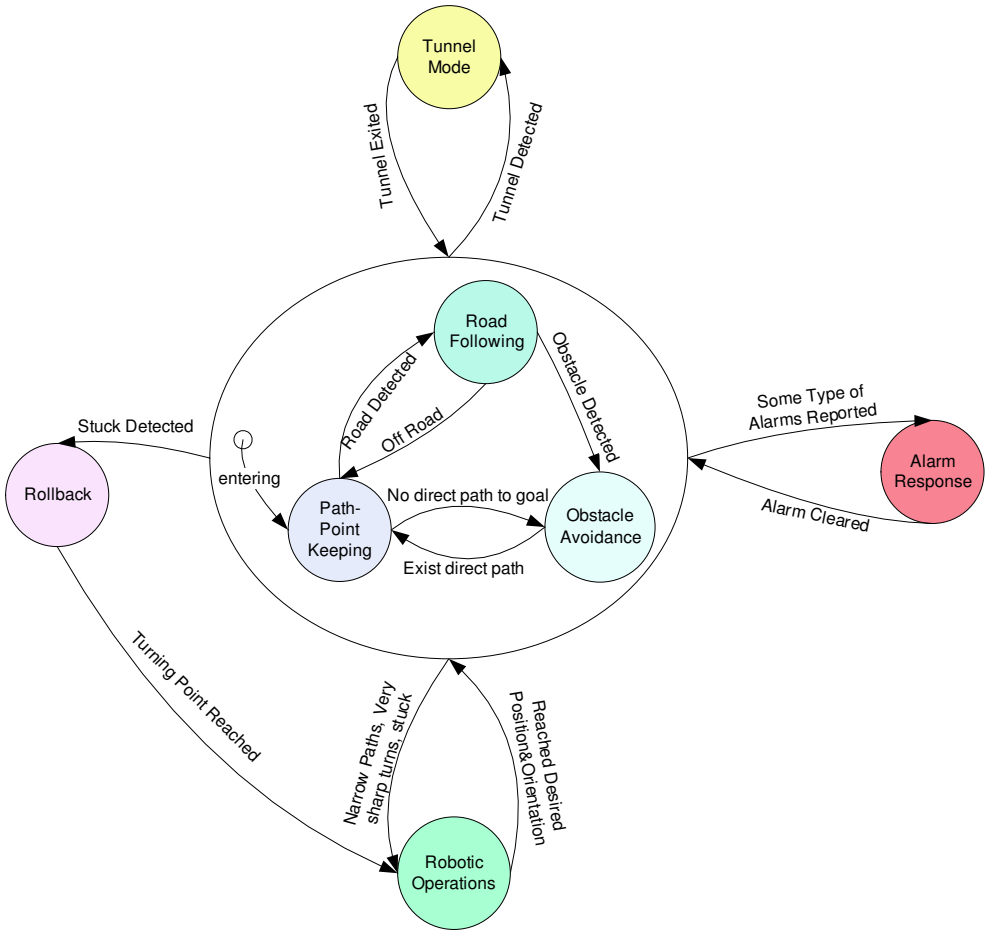


Figure 3.5: The finite state machine in ION's navigation module.

are generated by path-planning algorithms, which will be described in Chapter 4. A smooth path generation algorithm is used when the machine state is in the *path-point keeping* state. An obstacle-avoidance algorithm is used to generate obstacle-free local path when the machine state is in the *obstacle avoidance* state. The *road following* state is designed for the situation when the image-processing module detects a road[7]. The central meta-state starts from the *path-point keeping* state, both at the initialization and when machine state switches back from other states.

[7]Road following module was developed but not implemented in ION for the DGC05 race.

In the FSM, the other states are also very essential to deal with special events that the vehicle may encounter. The *tunnel mode* state is designed specifically for the situation when the AGV is in a tunnel where GPS signals are lost temporarily. We assume ION encounters no obstacles in the tunnels. In the *tunnel mode* state, the distances from the vehicle to the walls in both sides are measured and used to keep the vehicle in the center of the tunnel. The FSM switches into the *robotic operations* state when ION needs to adjust its heading or position or both, for example, when narrow paths and very sharp turns, i.e. direct forward path would contact obstacles, are encountered. In this state, a sequence of back-and-forth operations is executed to adjust ION's status. When there are sensor failures, the FSM transits into the *alarm* state to deal with the malfunction. Basically, while in this state, the navigator pauses the vehicle and waits for the sensor recovery[8]. In the case that the sensor failure is unrecoverable, a flag is received from the sensor fusion module and the FSM switches back to the central meta-state. Carrying these flags, the navigation module reduces the maximum allowed speed to reduce the risk of collision, and the sensor fusion module adjusts the coefficients correspondingly as well. The FSM state switches to the *rollback* state when the vehicle remains at the same site and cannot achieve reasonable progress for a long time. In this state, ION drives back along the trajectory it records till the point from where another path can be initiated. The *rollback* state design could be useful when there is more than one path choice at certain decision points[9].

[8]The sensory processing module detects the sensor failure to the supervisory module and promptly initiates the sensor resetting procedure to recover the failed sensor

[9]However, the race routes defined by DARPA in DGC05 are too narrow for such kind of decision points to exist. Therefore, the *rollback* state was not fully implemented in ION for the DGC race.

We also introduce a *watchdog* to prevent the vehicle from being stuck forever. The vehicle might be stuck when the *robotic unit operations* couldn't adjust vehicle status amidst obstacles in six tries, or false obstacles block the path totally, or because of unexpected events such as vehicle wheels being trapped. When ION's position does not change or the FSM rests in a state other than the central meta-state for a certain period of time, the FSM automatically resets to the *path-point keeping* state and stays in this state regardless of the obstacles and other events until ION reaches a certain distance. During the FSM resetting process, a *robotic unit operation* command with high throttle value is sent to the lower level controller in order to run over the obstacles, such as small bushes. With the watchdog design, ION might be able to get out of stuck in many cases and gain chances to continue.

CHAPTER 4

A REAL-TIME NAVIGATION ALGORITHM FOR AUTONOMOUS GROUND VEHICLES

4.1 Introduction

In this chapter, we will describe a *real-time obstacle avoidance algorithm* that is implemented in AGV navigation modules, also known as navigators. As described in the generic architecture in Chapter 2, the fundamental task of the navigator is to guide the AGV toward the goal without collision with obstacles. The obstacle avoidance algorithm presented in this chapter is one of the path-planning algorithms that generates safe and goal approaching local paths in real-time.

4.1.1 Concerns on the Navigation Module

There is a key design issue to fulfill the above requirements in AGV control systems. That is to deal with high complexity and rich details in short time and with limited computing resources.

The navigator is responsible for analyzing the situations, planning local paths, and dealing with discrete events. In addition, the navigator has to run at a short time scale so that it can respond promptly to the environment changes. Therefore, obstacle avoidance algorithms that require extreme computation are not suitable for

fast-moving AGVs with limited computation resources. Trade-offs should be made between rich information utilization and quick reaction capability.

As described in Chapter 3.3, the strategy used in ION's navigator is the discrete-event state machine. By perceiving both the environment changes and the AGV's status changes from the world model data, the state machine activates a proper drive mode to generate suitable commands. For example, in ION, the navigator plans or replans the local path when the machine state is at the center meta-state. The navigator checks the status of the local path that consists of 10 GPS points, each of which is called *pathpoints*. If the local path comes across obstacles in the local map, or the AGV reaches the 5th pathpoint of the local path, or a required waypoint is passed, the navigator generates a new local path. Figure 4.1 shows the procedure to generate the local path. There are two path-generating algorithms implemented in the navigation module: the *smooth-path algorithm* and the *obstacle avoidance algorithm*. The former plans smooth paths that concerns the vehicle kinematics (e.g. the minimum turning radius at the nominal velocity), which will be discussed in Chapter 5. The latter is used to generate an obstacle-free path to the goal.

As defined in the generic architecture, a navigator is the navigating controller in the tactical level. Therefore, the navigator's task is to generate a local path and a speed setpoint. It is assumed that the skill controllers can control the AGV to follow the reference pathpoints at the specified speed setpoint with acceptable bounded errors. Furthermore, the world model data is frequently updated so that it can provide the navigator with the ego-states of the AGV, such as 3D-position, speed, acceleration, yaw, roll, pitch, and their rate, etc., and the environment information around the vehicle, such as the position and shape of the nearby obstacles. Since the world

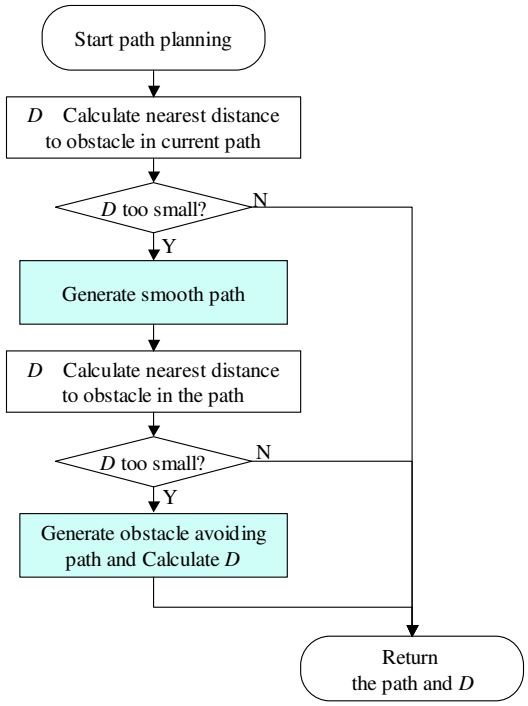


Figure 4.1: The flow chart for path planning.

model inevitably contains noise information, apart from its defined functionalities such as path planning and situation analysis, the navigator should also have certain fault tolerance capability.

4.1.2 Concerns on Obstacle Avoidance Algorithms

Since the late 1970s, extensive effort has been paid to develop obstacle avoidance algorithms, see literature reviews in Section 1.2 and 1.3. The research in this field can be divided into two major areas [77]: the global path planning [78, 79, 80] and the real-time local motion planning [81, 82, 83, 44].

The global path-planning methods, in general, assume that a complete global model of the environment with obstacles is known precisely. For an AGV, however,

it is unlikely to be able to precisely obtain the global information of the environment. Usually, it is only able to sense the local obstacles, which constraints the applications of these algorithms in the navigation module. They are more suitable to be implemented in the *strategic controller* to generate (globally) optimal plan according to certain criteria and currently available data. When the distances to the goal is comparable to the resolution of prior known data, the global path-planning method might not be used. For example, in DGC04 and DGC05, the race routes were described with road definition data that specified the GPS coordinates, width and speed limit for each section. The provided data was quite dense and the global path-planning methods were not needed in those events.

On the other hand, the local motion planning methods dynamically guide the AGV according to the locally sensed obstacles, which requires less prior knowledge about the environment. Usually, a feasible, instead of optimal, path is found by local motion planning methods. But they are more suitable and practical for the fast-moving AGVs, since the environment is probably unknown in prior and may be time-varying.

The concept of fuzzy set theory was presented by Zadeh in 1965 [84]. Since then, fuzzy logic has been successfully applied in many areas such as fuzzy control, decision making, approximate reasoning, etc. Fuzzy control is one of the most successful applications. Several obstacle avoidance algorithms based on fuzzy logic have been proposed and implemented in small robots, rovers and AGVs with small turning radius, see [85, 86, 41, 42, 87, 43, 88] and references therein. Most of these approaches select the direction by weighting the effort of *target-approach* and the need of *obstacle-avoidance*. As far as large AGVs like ION are concerned, few fuzzy logic approaches

consider the AGV's certain dynamic and kinematics constrains. For example, if the two consecutive steering decisions are totally opposite, such as left turn and right turn, the steering decisions will neutralize with each other. Therefore, frequent jumps between two consecutive steering decisions should be avoided. Also, many approaches do not respond to obstacles before being too close to them, so that the local path may be infeasible for the AGVs with considerably large turning radius.

In the remainder of this chapter, we will present a real-time local motion planning algorithm utilizing fuzzy logic that is suitable for large AGVs with dynamic and kinematics constraints. The obstacle avoidance algorithm consists of two fuzzy controllers: the *fuzzy steering controller* and the *fuzzy speed controller*[10]. The former focuses on planning goal-reaching and obstacle-free paths, and the latter focuses on setting proper speed setpoints.

The notations used in this chapter are listed in Table 4.1.

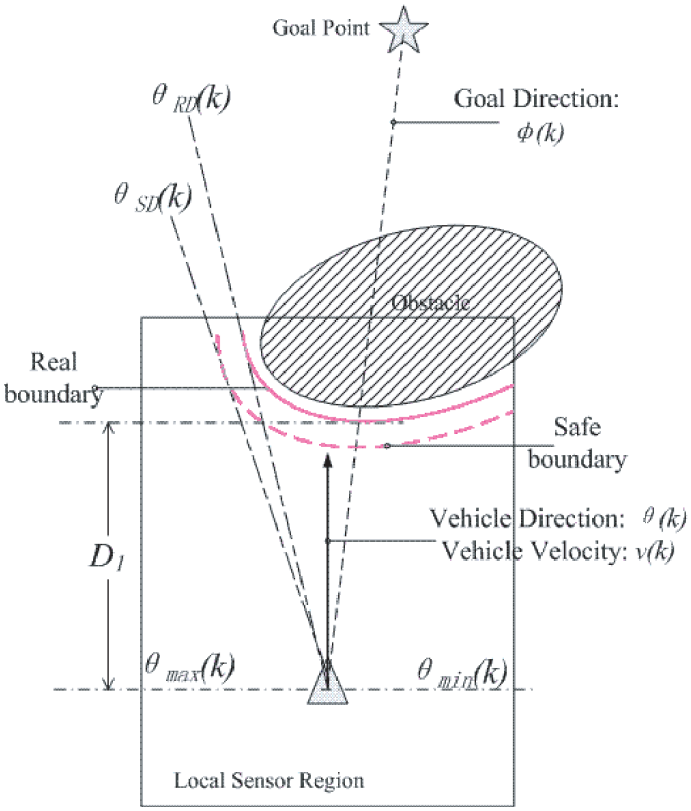
4.2 Fuzzy Steering controller

Figure 4.2 shows one example scenario that explains the basic steering algorithm. Based on the local sensor ability, the boundary of the obstacle in the local sensor region is detectable. Practically, we expand the boundary of the obstacle by half the size of the vehicle so that we can consider the vehicle as a mass point. The expanded obstacle boundary is called the *real boundary*, henceforth. Furthermore, to ensure the safety of ION, the obstacles are further enlarged to create a *safe boundary*, so that the real boundaries are enveloped with the safe boundaries.

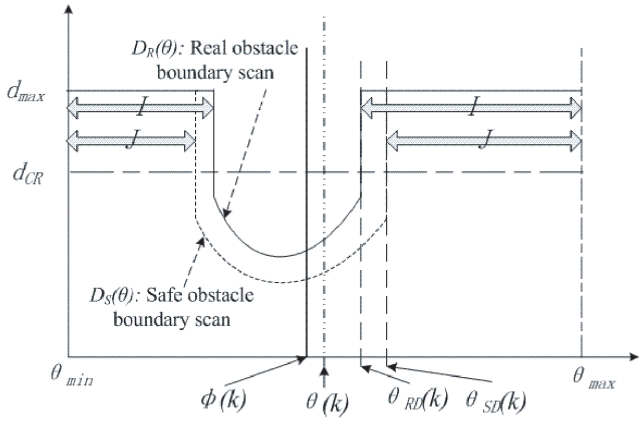
[10]These names have been used to indicate the *skill level controllers* in other chapters. In this chapter, the same names are used to indicate the controllers at the *tactical level*.

$\theta(k)$	the direction of the vehicle at the time k
$v(k)$	the speed of the vehicle at the time k
$\phi(k)$	the goal direction at the time k
$D_R(\theta)$	the distance from the vehicle to the real-obstacle-boundary in the direction θ
$D_S(\theta)$	the distance from the vehicle to the safe-obstacle-boundary in the direction θ
$\theta_{RD}(k)$	the decision based on the real-obstacle-boundary at the time k
$\theta_{SD}(k)$	the decision based on the safe-obstacle-boundary at the time k
$\theta_D(k)$	the final decision direction at the time k
$d_{\max}$	the maximum sensing distance
$\theta_{\min}(k)$	the start angle of the obstacle scanning, for example, $\theta(k) - \frac{\pi}{2}$
$\theta_{\max}(k)$	the end angle of the obstacle scanning, for example, $\theta(k) + \frac{\pi}{2}$
$\mathcal{I}$	the candidate set of $\theta_{RD}(k)$ at the time k
$\mathcal{J}$	the candidate set of $\theta_{SD}(k)$ at the time k
$d_{CR}(\theta)$	the minimum distance-to-obstacle for direction θ to be selected, used in $\mathcal{I}$ and $\mathcal{J}$
D_{safe}	the minimum safe distance-to-obstacle in front of the vehicle
D	the nearest distance to obstacle in the path, its value is calculated in the path-planning procedure, as in Figure 4.1
D_1	the distance to the obstacle in front of the vehicle, which is equivalent to $D_R(\theta(k))$
V_0	the current speed limit, a reference value
$V_s(k)$	the speed setpoint that the speed controller generates
F_s	the choosing strategy in choosing the $\theta_D(k)$
F_d	the fuzzy strategy in deciding the $d_{CR}(\theta, k)$
v_A	the speed setpoint generated for anticollision purpose
T	the safe-turning strategy for speed control

Table 4.1: The notation list



a. A local navigation scenario.



b. The scanning distances to the obstacle's boundaries at time k.

Figure 4.2: An example scenario. In the figures, $[\theta_{\min},\ \theta_{\max}]$ is the scanning angle range of interest; θ_{RD} and θ_{SD} are the decision angle based on the real boundary and safe boundary, respectively. D_1 is the distance to the obstacle in the current heading direction. d_{CR} is the criterion distance. $d_{\max}$ is the maximum scanning distance.

By the two-step obstacle extension, each obstacle has two different boundaries, the *real boundary* and the *safe boundary*. The distances to these boundaries are scanned and stored in two arrays: the real-boundary-distance array $D_R(\theta)$ and the seal-boundary-distance array $D_S(\theta)$, where $\theta \in [\theta_{\min}(k),\ \theta_{\max}(k)]$ is the direction of interest. Figure 4.2b illustrates the boundaries and the scanning distances.

The fuzzy steering controller described in this section originates from a heuristic non-fuzzy algorithm that has deficiencies in certain situations. Fuzzy rules are developed to deal with the shortcomings so that the improved algorithm fits in most situations that an AGV might encounter.

4.2.1 Basic Steering Controller

The basic steering strategy is to select the direction that has the smallest angle difference to the goal direction. The formulas are expressed as follows:

$$\begin{aligned} \mathcal{I}(k) &= \{\theta | D_R(\theta) > d_{CR}\} \\ \mathcal{J}(k) &= \{\theta | D_S(\theta) > d_{CR}\} \end{aligned} \tag{4.1}$$

and

$$\begin{aligned} \theta_{RD}(k) &= \arg \min_{\theta \in \mathcal{I}(k)} \{|\theta - \phi(k)|\} \\ \theta_{SD}(k) &= \arg \min_{\theta \in \mathcal{J}(k)} \{|\theta - \phi(k)|\} \end{aligned} \tag{4.2}$$

where $\mathcal{I}$ and $\mathcal{J}$ are the candidate sets of angles at which the distance-to-obstacle (DTO) is larger than d_{CR}. d_{CR} is a design parameter in this basic steering controller and $d_{CR} < d_{\max}$.

Equations (4.1) and (4.2) provide the method to find two directions, $\theta_{RD}(k)$ and $\theta_{SD}(k)$, as decisions based on the real boundary and the safe boundary scan distances, respectively. The final decision direction, $\theta_D(k)$, is then selected from these two directions. We may prefer to choose $\theta_{SD}(k)$ because the direction selected according

$\|\theta_{SD}(k) - \phi(k)\|$ \ $\|\theta_{SD}(k) - \theta_{RD}(k)\|$			
$\theta_D(k) =$	Small	Medium	Large
Small	$\theta_{SD}(k)$	$\theta_{SD}(k)$	$\theta_{RD}(k)$
Medium	$\theta_{SD}(k)$	$\theta_{RD}(k)$	$\theta_{RD}(k)$
Large	$\theta_{SD}(k)$	$\theta_{RD}(k)$	$\theta_{RD}(k)$

Table 4.2: The selection rule

to the safe boundaries guarantees a certain safe distance from contacting obstacles. However, since the safe boundaries are expanded more than the real boundaries, it may block a narrow path with is passable for the vehicle. In this case, $\theta_{RD}(k)$ is a better direction to be selected than $\theta_{SD}(k)$ is.

The selection rule is shown in Table 4.2, where the "Small", "Medium" and "Large" are fuzzy descriptions. By comparing the angle distance between $\theta_{RD}(k)$, $\phi(k)$, and $\theta_{SD}(k)$, the defuzzification function picks either $\theta_{SD}(k)$ or $\theta_{RD}(k)$, as the result of $\theta_D(k)$. This ensures that the result would not be a value between $\theta_{SD}(k)$ and $\theta_{RD}(k)$.

Equation (4.3) represents the selecting procedure:

$$\theta_D(k) = F_s(\theta_{RD}(k), \theta_{SD}(k), \phi(k)) \tag{4.3}$$

Remark 1 *If $\theta_{SD}(k)$ is chosen to be the steering decision, then there is an open space in the direction so that vehicle is allowed to pick up a relatively high speed safely. On the other hand, if $\theta_{RD}(k)$ is chosen, it can be concluded that $|\theta_{SD}(k) - \theta_{RD}(k)|$ is large, i.e. the $\theta_{SD}(k)$, presenting the open free space, is in the other direction, and*

the chosen direction must be a narrow path. The speed controller uses this information and sets a relatively low speed setpoint.

Property 4.1 *If there is no obstacles between the vehicle and the goal, i.e.* $D_S(\phi(k)) = d_{\max}$*, then* $\theta_D(k) = \phi(k)$*.*

Proof: Since $D_S(\phi(k)) = d_{\max}$, then $\phi(k) \in \mathcal{J}(k)$, which implies

$$\theta_{SD}(k) = \arg \min_{\theta \in \mathcal{J}(k)} \{|\theta - \phi(k)|\} = \phi(k)$$

Similarly, we can conclude $\theta_{RD}(k) = \phi(k)$ too, since the the real boundaries are enveloped with the safe boundaries, i.e. $D_R(\theta) \geq D_S(\theta)$.

Combining the above gives that $\theta_D(k) = \phi(k)$. ■

The criterion distance, d_{CR}, in the equations (4.1) is an important parameter in this basic steering controller. If d_{CR} is carefully selected, the basic fuzzy steering controller may work well when there are few obstacles in the environment and the obstacles are all convex-shaped.

However, this is not the case for general AGV environments. When the environment has many obstacles or the obstacles are concave-shaped, there exist four major problems in this basic steering controller:

- First, if d_{CR} is chosen too large, then both $\mathcal{I}$ and $\mathcal{J}$ could be *empty* sets when the vehicle is surrounded by obstacles. On the other hand, if d_{CR} is chosen too small, then both $\mathcal{I}$ and $\mathcal{J}$ equal to the interval $[\theta_{\min} \;\; \theta_{\max}]$, and it implies that $\theta_D(k) = \theta_{SD}(k) = \theta_{RD}(k) = \phi(k)$. In other words, there is no obstacle avoidance at all until the vehicle is very close to an obstacle.

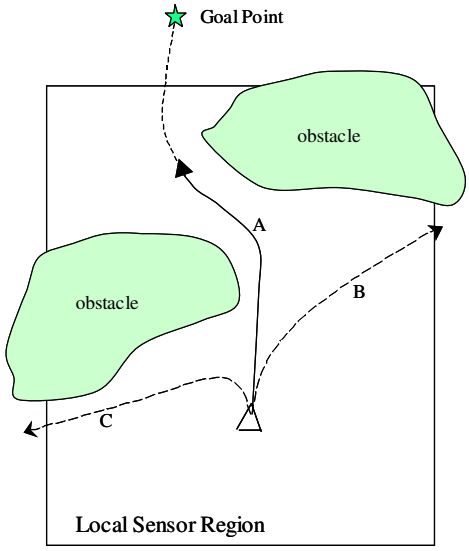


Figure 4.3: When d_{CR} is chosen large, the basic steering strategy chooses path B or C instead of path A.

- Second, the basic steering controller sometimes makes wide-left turn (or wide-right turn) instead of finding a path in the front just because there is no obstacles in there. The steering controller selects the wide-left direction because the DTO in the left equals $d_{\max}$ and there are obstacles in other directions. This may result in inefficiency or problem, for example, as shown in Figure 4.3, the steering controller would rather pick detour directions than find paths in its front.

- Third, the AGV may easily be *stuck* at certain concave-shaped obstacle surfaces. This happens when d_{CR} is small so that the steering controller does not try to avoid the obstacle until gets into the concave-shaped trap.

- Fourth, the navigator may generate *discontinuous* output of $\theta_D(k)$, i.e. $|\theta_D(k) - \theta_D(k-1)|$ is large. There is no constraint applied on the changes between two

consecutive decisions. We may allow big changes in steering decisions once in a while in order to deal with the sudden changes in the environment. Frequent large changes in the decisions, however, are not acceptable because it may inject disturbance to the skill controllers. In most cases, the last two problems happen together and cause the AGV to go toward an obstacle and get stuck.

We propose the following rules to solve all of above problems, so that our steering controller can deal with most situations and guide the AGV to the goal safely.

4.2.2 Focus Rule

Focus rule is introduced in the steering controller in order to avoid selecting wide-left (or wide-right) directions while there are feasible paths in the AGV's front, such as the situation shown in Figure 4.3. In the *focus rule*, we consider the directions pointing forward and leading to the goal point to have higher priorities than others.

The formulas representing the *focus rule* is presented in the following:

$$\begin{aligned} \mathcal{I}(k) &= \{\theta | D_R(\theta) - \Delta(\theta) > d_{CR,R}(k)\} \\ \mathcal{J}(k) &= \{\theta | D_S(\theta) - \Delta(\theta) > d_{CR,S}(k)\} \\ d_{CR,R}(k) &= \min\{d^0_{CR,R}, \max_\theta\{D_R(\theta) - \Delta(\theta)\} - \epsilon\} \\ d_{CR,S}(k) &= \min\{d^0_{CR,S}, \max_\theta\{D_S(\theta) - \Delta(\theta)\} - \epsilon\} \end{aligned} \tag{4.4}$$

where the candidate sets, $\mathcal{I}$ and $\mathcal{J}$, replace the ones defined in (4.1). $\Delta(\cdot)$ is introduced to reshape the DTO curves, $D_R(\cdot)$ and $D_S(\cdot)$. When a direction θ has low priority, the $\Delta(\theta)$ is then set to a large value so that the direction θ is less likely to be in the candidate sets, $\mathcal{I}$ and $\mathcal{J}$. By doing so, the steering controller tends to find the path direction with high priority. $d^0_{CR,R}$ and $d^0_{CR,S}$ are design parameters to prevent the algorithm from being sensitive to obstacles very far away where the probability of

noisy information, especially false obstacles, is high, due to low sensor resolution and sensor coverage. ϵ is a small positive constant so that the candidate sets are both nonempty.

Define $\Delta 1(\theta) = |\theta - \theta(k)|$ and $\Delta 2(\theta) = |\theta - \phi(k)|$, where $\theta(k)$ is the vehicle heading at time k. So, a direction with high priority has small values of $\Delta 1$ and $\Delta 2$. Let (S, MS, M, ML, L) represent (*Small, Medium Small, Medium, Medium Large*, and *Large*), respectively. The fuzzy *focus rule* that decides the value of $\Delta(\theta)$ in the equation (4.4) is as follows:

- If ($\Delta 1$ is S) and ($\Delta 2$ is S), then (Δ is S): if θ points forward and leads to the goal direction, then θ has the highest priority to be chosen, i.e. $D_R(\theta)$ or $D_S(\theta)$ has the *smallest* deduction.

- If (($\Delta 1$ is S) and ($\Delta 2$ is M)) or (($\Delta 1$ is M) and ($\Delta 2$ is S)) or (($\Delta 1$ is M) and ($\Delta 2$ is M)), then (Δ is MS).

- If (($\Delta 1$ is S) and ($\Delta 2$ is L)) or (($\Delta 1$ is L) and ($\Delta 2$ is S)), then (Δ is M).

- If (($\Delta 1$ is L) and ($\Delta 2$ is M)) or (($\Delta 1$ is M) and ($\Delta 2$ is L)), then (Δ is ML).

- If ($\Delta 1$ is L) and ($\Delta 2$ is L), then (Δ is L): if the angle θ neither points forward nor leads to the goal direction, it has low priority and the deduction is *large*.

Property 4.2 *If $\Delta(\theta) \geq D_R(\theta)$ for some θ, then $\theta \neq \theta_D(k)$.*

Proof: If $\Delta(\theta) \geq D_R(\theta)$ for some θ, then according to (4.4), $\theta \notin \mathcal{I}$. Since the real boundaries are enveloped with the safe boundaries, i.e. $D_R(\theta) \geq D_S(\theta)$, then $\Delta(\theta) \geq D_S(\theta)$, so that $\theta \notin \mathcal{J}$ too. Thus, $\theta \notin \mathcal{I} \cup \mathcal{J}$.

On the other hand, according to (4.2), $\theta_{RD}(k) \in \mathcal{I}$ and $\theta_{SD}(k) \in \mathcal{J}$. Also, according to the selection rule, $\theta_D(k) \in \{\theta_{RD}(k),\ \theta_{SD}(k)\}$, so we have $\theta_D(k) \in \mathcal{I} \cup \mathcal{J}$.

Combining the above gives $\theta \neq \theta_D(k)$. ■

Property 4.2 shows that for a direction θ, if $\Delta(\theta) \geq D_R(\theta)$, then the direction is excluded from being selected as the decision direction. By doing so, the steering controller opens a "window" for those directions with high priority. Thus, the steering controller ignores the distractions from low priority directions, such as the wide-left (or wide-right) directions. A direction with low priority is selected only if the directions with higher priority are all blocked by obstacles.

Figure 4.4 shows a simulation example of how the *focus rule* works. The "world model map" is shown in the upper left figure and the obstacle boundaries are shown in the upper right figure. According to the original scanned DTOs, shown in the middle figure, the decision without using *focus rule* would lead the vehicle to the very right side of the obstacles, though there is a better passable path in the middle. By applying the *focus rule*, the $-\Delta(\cdot)$ curve is Λ-shaped if $\theta(k)$ is close to $\phi(k)$. According to the modified DTO curves ($D_R(\theta) - \Delta(\theta)$ and $D_R(\theta) - \Delta(\theta)$) plotted in the bottom figure, the selected direction $\theta_D(k)$ is obtained. The selected direction is plotted in the upper right figure, following which the AGV avoids the obstacles and approaches the goal point more efficiently than without using the *focus rule*.

In this fuzzy *focus rule*, triangular membership functions are used to define the fuzzy sets for the $\Delta 1$ and $\Delta 2$, as shown in Figure 4.5. The vales assigned to ($S,\ MS,\ M,\ ML,\ L$), for the fuzzy sets of Δ, determine the height of the hump. The larger are the values, the higher the hump is. The values of (a, b, c, d) determine the membership function values for $\Delta 1$ and $\Delta 2$, consequently determine the width

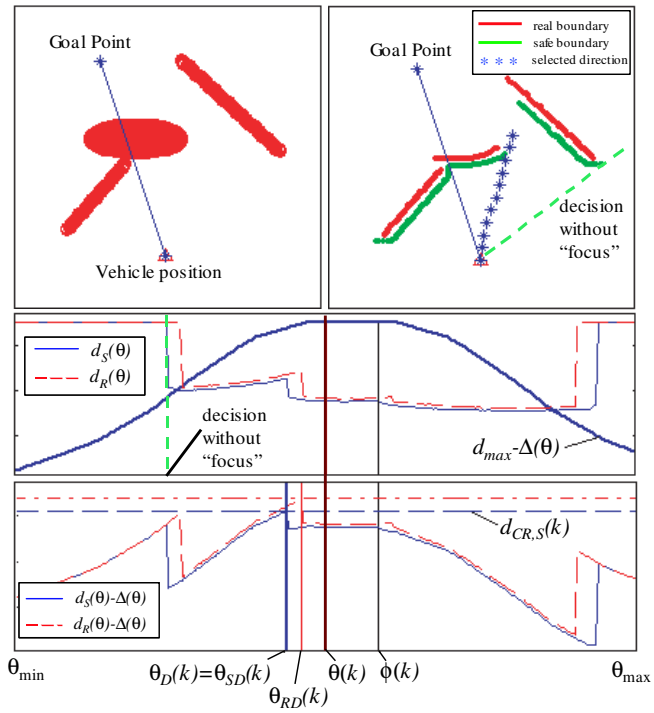


Figure 4.4: The example of the *focus rule*. The upper left figure is the "world model map" and the upper right figure displays the obstacle boundaries. The scanned DTO curves, $D_R(\theta)$ and $D_S(\theta)$, are shown in the middle figure. The $d_{max} - \Delta(\theta)$ is a Λ-shaped curve plotted in the middle figure, and the curves of $D_R(\theta) - \Delta(\theta)$ and $D_R(\theta) - \Delta(\theta)$ are plotted in the bottom figure. The finial direction θ_D is then selected according to equation (4.2) and the selection rule represented in Table 4.2. θ_D is shown in the upper right figure, following which the AGV avoids the obstacles and approaches the goal point.

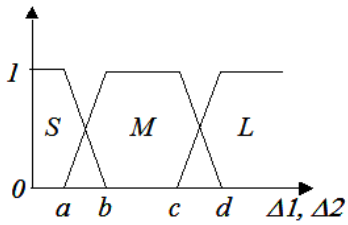


Figure 4.5: The membership functions for $\Delta 1$ and $\Delta 2$.

of the hump in the curve of $-\Delta(\theta)$. If the values are small, then $\Delta 1$ and $\Delta 2$ easily belongs to "Large" so that Δ tends to be large. As the result, the DTO are reduced in most directions, leaving a narrow range of direction with unreduced DTO. Thus, the focus effort is enhanced. On the other hand, if the values of (a, b, c, d) are large, then the focus effort is neutralized and the navigator has a wide range of direction to select from.

4.2.3 Focus-vs-Search Rule

The *focus rule* avoids the distractions by side directions quite well. Nevertheless, the *focus rule* trades the ability in searching the feasible directions for the distraction prevention. It works well when there are few obstacles in the environment. When there are many obstacles in the environment, however, the *focus rule* impairs the search capability of the navigator.

Furthermore, as stated previously, the values of (a, b, c, d) in the membership function of the fuzzy sets of $\Delta 1$ and $\Delta 2$ are important parameters of the *focus rule*. Smaller values of (a, b, c, d) are corresponding to stronger "focus", and vice versa. Motivated by this, the *focus rule* is improved by considering the fuzzy evaluation of the AGV status. The $v(k)$, the vehicle's speed, is regarded as the index of vehicle's situation: the lower the speed is, the more search capability is required.

Therefore, the equations are:

$$\begin{aligned} S(k) &= F_S(v(k), V_0) \\ [a(k)\ b(k)\ c(k)\ d(k)]^T &= F(S(k)) \end{aligned} \tag{4.5}$$

where $S(k)$ is the situation classification, a membership function values of the fuzzy sets: *Exploring (EX)* and *Travelling (TR)*.

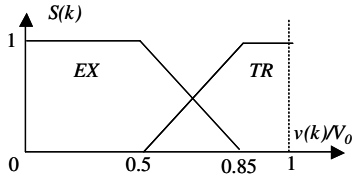


Figure 4.6: The membership functions for the situation classification $S(k)$.

The membership function is shown in Figure 4.6. The rule $F_S(\cdot)$ states: When the $v(k)/V_0$ is small (close to 0), $S(k)$ belongs to fuzzy set *exploring* (EX). On the other hand, when $v(k)/V_0$ is large (close to 1), $S(k)$ belongs to fuzzy set *travelling* (TR). The AGV speed, $v(k)$, is classified by comparing rather to the base speed rather than by the absolute values. The base speed, V_0, is provided to the AGV as the current speed limit. In the *focus-vs-search rule*, the values of (a, b, c, d) are calculated. The fuzzy rule in (4.5) is as follows:

- If $S(k)$ is EX, then the values of (a, b, c, d) increases: When the AGV is closing to an obstacle, we assume that the speed controller slows down the AGV. In this case, the $S(k)$ becomes more "EX", so that the values of (a, b, c, d) increases. Thus, the *focus rule* is neutralized and the "search" capability is enhanced.

- If $S(k)$ is TR, then the values of (a, b, c, d) decreases: vice versa, the *focus rule* is enhanced.

By applying the *focus-vs-search rule*, the AGV is much less likely to be stuck in front of the concave-shaped obstacles. When the AGV approaches an obstacle, the speed is reduced so that the "priority window" is opened up for search and the "focus" effect is neutralized.

4.2.4 Persistence Rule

The last problem is about *discontinuous* outputs from the steering controller, i.e. $|\theta_D(k) - \theta_D(k-1)|$ is large. Frequent large changes in the decision directions, especially, are not acceptable because it may inject disturbance to the skill controllers. Also, it may cause the oscillation of the vehicle's heading. In the worst situation, the two consecutive steering decisions may neutralize each other.

There are two methods to solve the problem: to define fuzzy rules, or to define strict rules.

Fuzzy Persistence Rule:

Using fuzzy logic, the value of $\theta_D(k-1)$ is introduced to adjust $\Delta(\theta)$ in the *focus rule*. By doing so, the steering controller adjusts the searching window and raises the priority for the directions that are near to the previous decision, $\theta_D(k-1)$. Therefore, the steering controller maintains the persistence. The *persistence rule* weighs on the last decision and gives the controller a characteristic of persistence so that the zigzag performance of the AGV is prevented.

Strict Rules:

When there is no obstacle between the vehicle and the goal point, according to Property 4.1, the steering controller always selects $\psi(k)$. In this case, there is no frequent large change in the steering controller output.

On the other hand, when there are obstacles in direction $\phi(k)$, then the steering controller would generate a direction, which avoids the obstacles by left or by right. In order to avoid that the selected direction switches frequently between the left and

the right, we define a flag to indicate the current choice and persist the choice in the following choices. A set of crispy rules, called *strict rules*, is defined as follows.

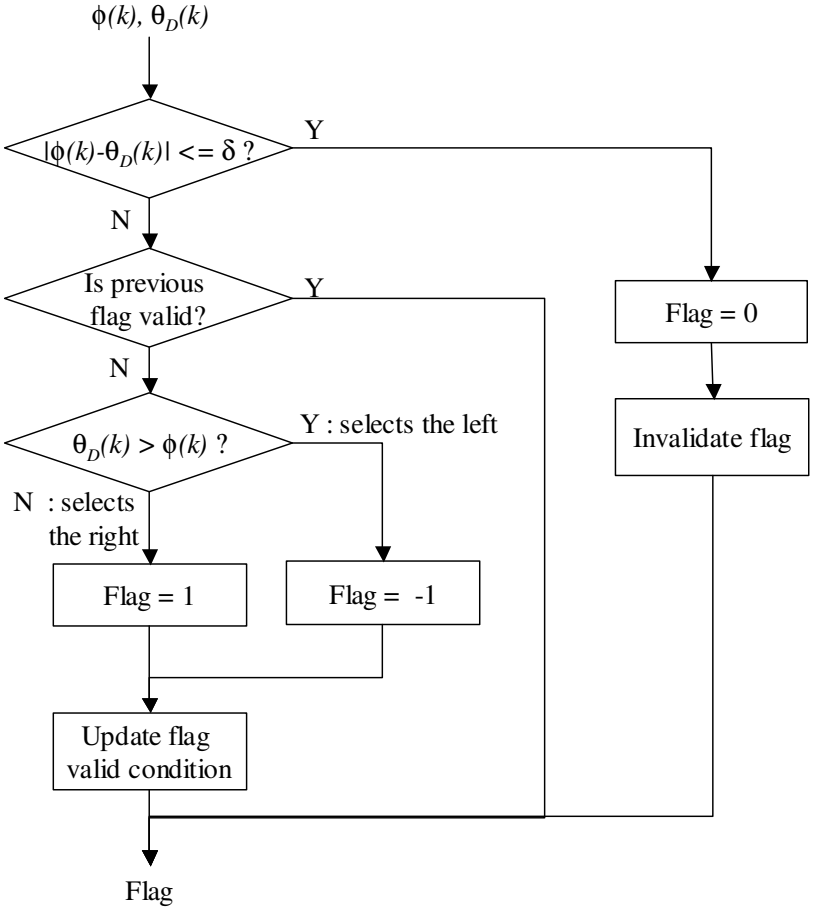
The strict persistence procedure consists of two parts: to update current choice and to enforce the current choice in the next direction selecting procedure.

As shown in Figure 4.7.a, the current decision direction $\theta_D(k)$ and the goal direction $\phi(k)$ are compared to update the flag that indicates the choice at time k. The flag value is updated if both of the following conditions are satisfied. First, $|\phi(k) - \theta_D(k)| > \delta$ for certain design parameter δ of small value, which means that there are obstacles between the vehicle and the goal point. The other condition is that the flag is invalid. The flag-valid-condition can be a certain time period since the latest flag update or a certain distance from where the latest flag has been updated, or both.

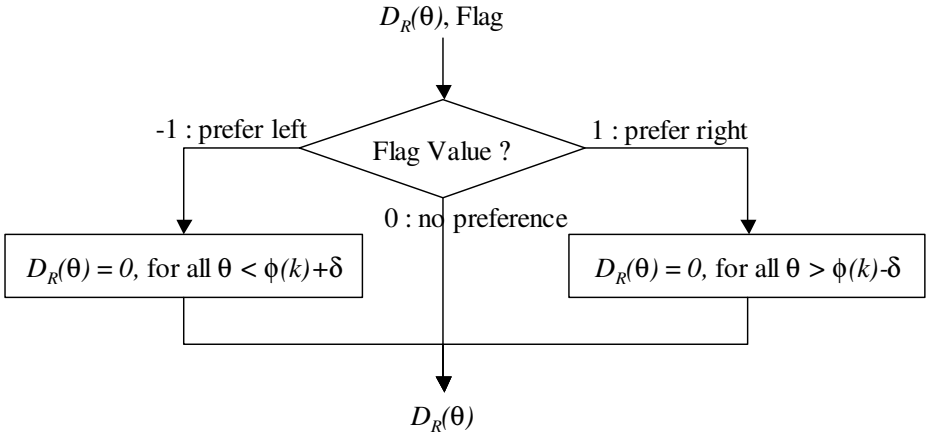
Figure 4.7.b shows how to use the choice flag in the next direction selecting procedure. When there is a valid flag, it is used to modify the DTO array, $D_R(\theta)$ (or $D_S(\theta)$) by setting the DTO values to 0 for the directions that contradict with the steering decision in the previous time. Thus, these directions would not be selected into candidate sets, $\mathcal{I}(k)$ or $\mathcal{J}(k)$, therefore, none of them would be the steering decision direction.

4.2.5 Brief Summary of the Fuzzy Steering Controller

Figure 4.8 summarizes the fuzzy steering controller described in this section. The input and output data of the rules indicated above are shown in this diagram.



a. Update the flag that indicates the choice, at time k .



b. Modify the DTO array according to the flag value, at time $k+1$.

Figure 4.7: The flowcharts of applying the strict persistence rule.

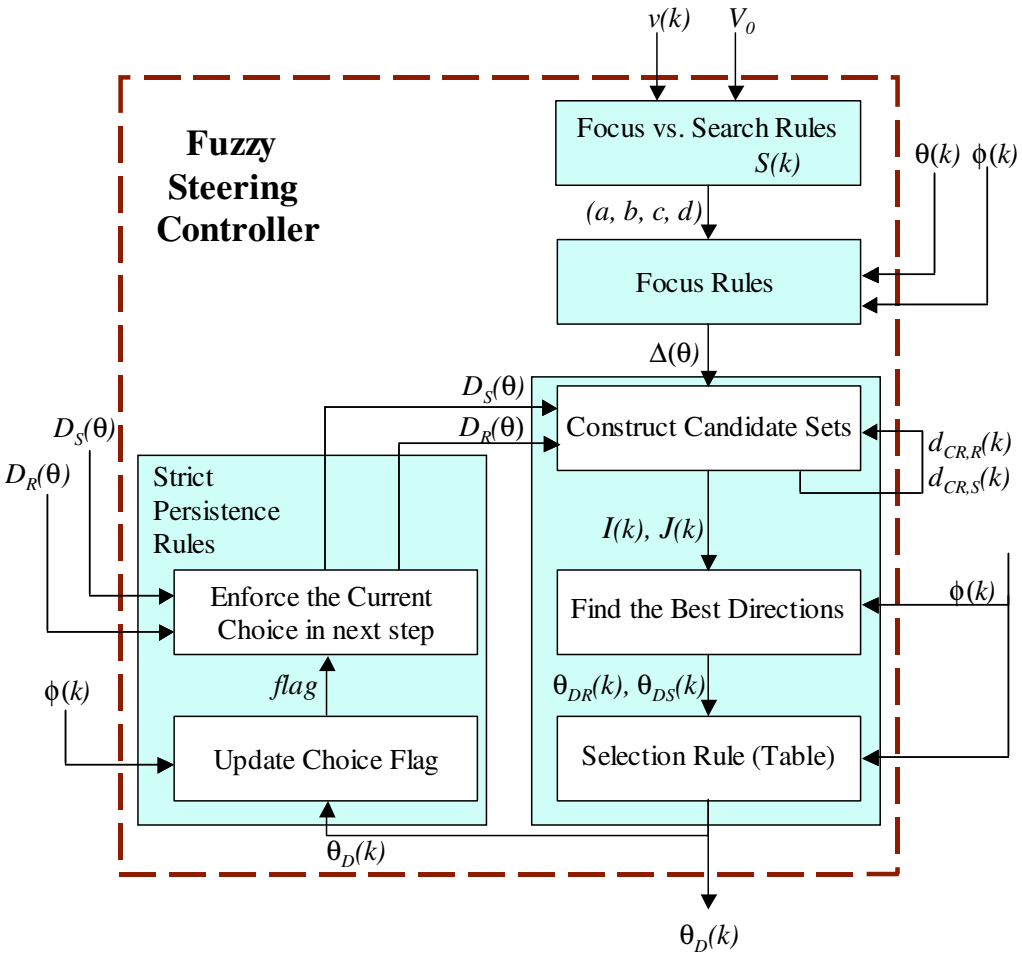


Figure 4.8: The structure of the fuzzy steering controller.

Like most of other local motion planning algorithms, the fuzzy steering algorithm uses only local information and the global optimality is not guaranteed. In the practical applications, when dealing with extreme difficult situations by using partial local information, the vehicle sometimes would get stuck. Figure 4.9 shows an example of a difficult situation for local motion planning algorithms, where there are N paths but only one of them leads to the goal point. For any AGV with maximum sensing distance of D, the distance it may travel before finding the right path is $2(N-1)(L-D)$

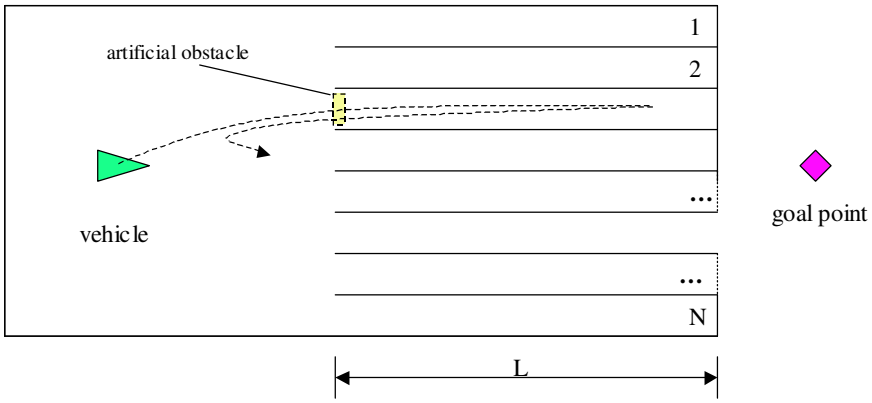


Figure 4.9: An example of difficult situation for local motion planning algorithms: N paths of length L with only one leads to the goal point and vehicle's maximum sensing distance is D, $D < L$. After the vehicle tried a path and got back to the initial position, an artificial obstacle is placed at the entrance of the path in the world model so that the path would not be repeated in the later tries.

in the worst case. This example also shows that local motion planning algorithms with maximum sensing distance D cannot avoid concave obstacles with depth of L, if $L > D$. There are many methods to deal with this stuck situation. For ION, when the vehicle is stuck, its state machine described in Section 3.3 switches into the *rollback* state or the *robotic operations* state so that the vehicle can drive back to the point where another path can be initiated by the navigator module. Furthermore, an artificial obstacle is placed at the entrance of the original path in the world model, as indicated in the figure, so that this path would not be repeated in the later tries.

In the practical applications, apart from applying the navigation algorithms in the steering controller, we also need to consider the data format of the world model map, which comes from the sensor and sensory process module. In ION, the detailed model map data is huge and needs to be relayed over network. In order to alleviate the network burden as well as obtain the accurate and recognizable map data, in ION,

we transfer the world model map data into scanned DTO arrays before sending it to the navigation module. The drawback, however, is that it may lose some information about the obstacles that are behind other obstacles. However, in some cases, the cost is bearable, because the detailed world model map data is huge and relaying these data from one computer to another is a heavy burden on the network. In order to alleviate the network burden as well as obtain the accurate and recognizable map data, in ION, we transfer the world model map data into scanned DTO arrays before sending to the navigation module. The drawback, however, is that it may lose some information about the obstacles that are behind other obstacles. However, in some cases, the cost is bearable, because:

- The scanned DTO arrays represent the front obstacles, which is closer (hence more important) than the obstacles in the behind;
- In many cases, the obstacles behind others are not reliable since they have few sensor coverage, low sensor resolution. They are probably measured several steps before (when it is viewable) and are polluted by cumulated noise;

Furthermore in our navigation module, we mark the regions behind obstacles as "unknown", which means it is not clear whether there are obstacles, since these regions have never been viewable. With DTO arrays, the number of directions in the arrays is of the designer's choice. It may be chosen according to how much computation resource is available. The larger the number is, the more computation time it requires, and the higher resolution it has in selecting the direction.

According to our experimental results, the above steering algorithms works in most situations.

4.3 Fuzzy Speed Controller

There are several concerns on selecting the speed setpoint. First, proper speed should be set to avoid the collision into the obstacles. Second, due to the physical constraint of the vehicle, sharp turning at high speed should be avoided to prevent the AGV from rolling over. The impacts of certain special discrete events on the speed setpoint should be concerned too.

Our strategy is to define the speed setpoints according to each of the concerns and then select the minimum one as the final speed setpoint. In the following, we discuss how the speed setpoints are calculated.

4.3.1 The Anticollision Rule

The main purpose of the *anticollision rule* is to prevent the vehicle from colliding into obstacles. Let v_A represent the speed setpoint generated according to the *anticollision rule*. The fuzzy rule have two inputs, D_1 and D, where D_1 represents the distance to obstacles in the front of vehicle and D is the value returned from path-planning procedure as in Figure 4.1. Let (S, MS, M, ML, L) have the same meaning as stated above, and let (DA) denote the fuzzy set of "dangerous distance". The rule is stated as follows:

- If (one of (D_1, D) is DA), then v_A is $STOP$: the vehicle should stop when the distance to the obstacles is too close and it is dangerous to move on.
- If (D_1 is S), then A is S: the vehicle speed should be reduced when it is close to the obstacles.
- If (D_1 is M and D is S), then v_A is MS.

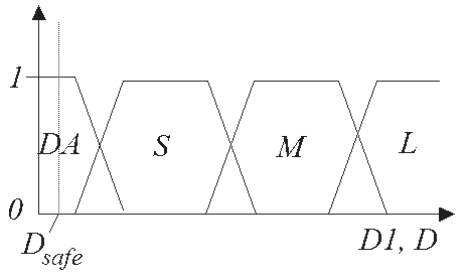


Figure 4.10: The membership functions for D_1 and D.

- If (D_1 is M and D is M), then v_A is M.
- If (one of (D_1, D) is L) and (the other is M), then v_A is ML.
- If (both of (D_1, D) are L), then v_A is L: no obstacle is nearby, the speed is set to a high value.

The boundaries of the fuzzy set are selected according to the dynamic performance of the vehicle and how much risk one wants to take. The membership functions of D_1 and D are shown in Figure 4.10. We use the same membership functions for D_1 and D are the same, although they can be different. Note that the D_{safe}, the minimum safe distance, is included in the fuzzy set DA with the membership value of 1, as indicated in the figure. Thus, before the AGV reaching the distance D_{safe}, the speed setpoint is set to 0, so that collisions are prevented. Should the AGV stop because of this (i.e. it is dangerously close to an obstacle), the higher level controller would detect the situation and switch the FSM state to *rollback* state or *robotic operation* state and wait to be recovered by a sequence of robotic operations.

4.3.2 Safe-Turning Rule

The safety in making a turn is also concerned. An AGV may roll over if the steering command is large (sharp turn) and the speed setpoint is high at the same time. More importantly, the path-following controllers in the *skill level* may overshoot at sharp turns, which may cause the collisions with obstacles even if the planned paths are obstacle-free.

Our experiments shows that reducing the speed setpoint at sharp turns can reduce the overshoot significantly and can shorten the distance for stabilizing the AGV heading. Thus, the *safe-turning rule*, denoted as T, is introduced. We define $\Delta\theta(k) = |\theta_D(k) - \theta(k)|$ to represent the steering command. Fuzzy logic has been used to define T: If ($\Delta\theta$ is S) then T is L; If ($\Delta\theta$ is M) then T is M; If ($\Delta\theta$ is L) then T is S. Or, T can be defined as a function mapping from $\Delta\theta$ to v_T, where v_T represents the speed setpoint generated for safe-turning purpose:

$$v_T = T(\Delta\theta)$$

The suitable mapping from $\Delta\theta$ to v_T can be established from repeated experiments so that it serves the safe-turning purpose well. Generally, $T : [0, \pi] \rightarrow [0, v_{\max}]$ is a decreasing function that may vary from one AGV to another, from one terrain type to another. As an example, Figure 4.11 shows the curve of the function T that we used for ION in desert-trail type of terrains.

Thus, the speed controller calculates the speed setpoint V_s as follows:

$$V_s(k) = \min[V_0(k), v_A(k), v_T(\Delta\theta(k)), v_E(k)] \tag{4.6}$$

where $V_0(k)$ is the speed limit. $v_A(k)$ and $v_T(k)$ are described in the above. $v_E(k)$ is the maximum speed allowance when the vehicle is under certain special conditions or

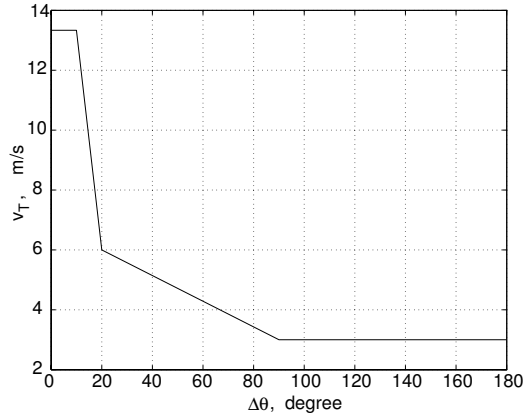


Figure 4.11: The mapping from $\Delta\theta$ to v_T for ION in desert.

when certain events occur. For example, in ION, we set $v_E(k)$ to small values when GPS signal is degraded, or the vehicle is still in the middle of a sharp turning, or the $\theta_{RD}(k)$ is chosen as the steering decision as mentioned in Remark 1. When there are no special conditions, $v_E(k)$ can be set to the maximum speed that the vehicle's hardware allows.

4.4 Performance of the Navigator

The navigation algorithm described in this chapter has been tested in simulation, in small robots, and in ION.

4.4.1 Simulation

A simulator has been built to test the navigation algorithm designed in this chapter. Figure 4.12 shows the user interface of the simulator. The "global map view" displays the global information including the mission plan (a series of waypoints), the prior known obstacles, the locally sensed obstacles, and the vehicle trajectory. The

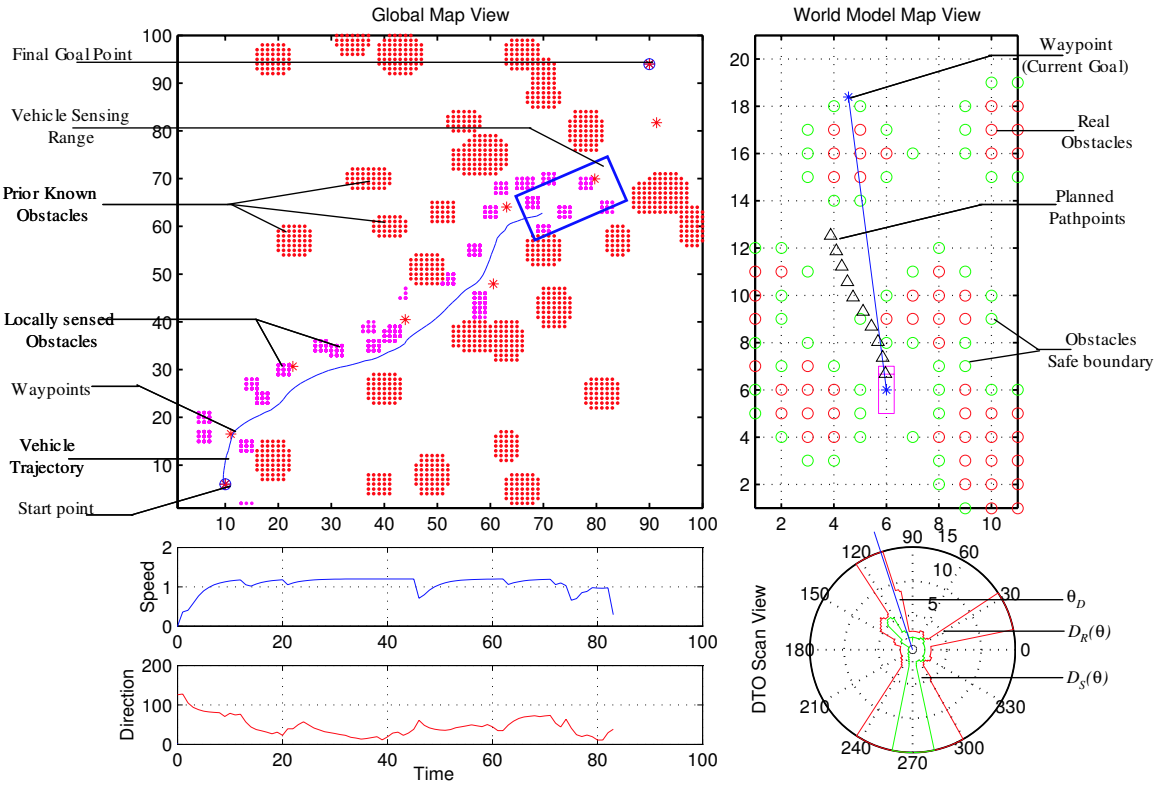


Figure 4.12: The simulator display.

"locally sensed obstacles" are detected by the vehicle and plotted on the map only when they are or have been in the simulated AGV sensing range. As a consequence, only those previously unknown obstacles that are near the vehicle trajectory are displayed in the result map as the "locally sensed obstacles". The "world model map view" displays both prior known or locally sensed obstacles that are in the AGV's sensing range, as indicated in the "global map view". The safe boundaries of the obstacles are displayed in the "world model map view" too. The scanned DTO arrays, both $D_R(\theta)$ and $D_S(\theta)$, are displayed in the "DTO scan view". The decision direction

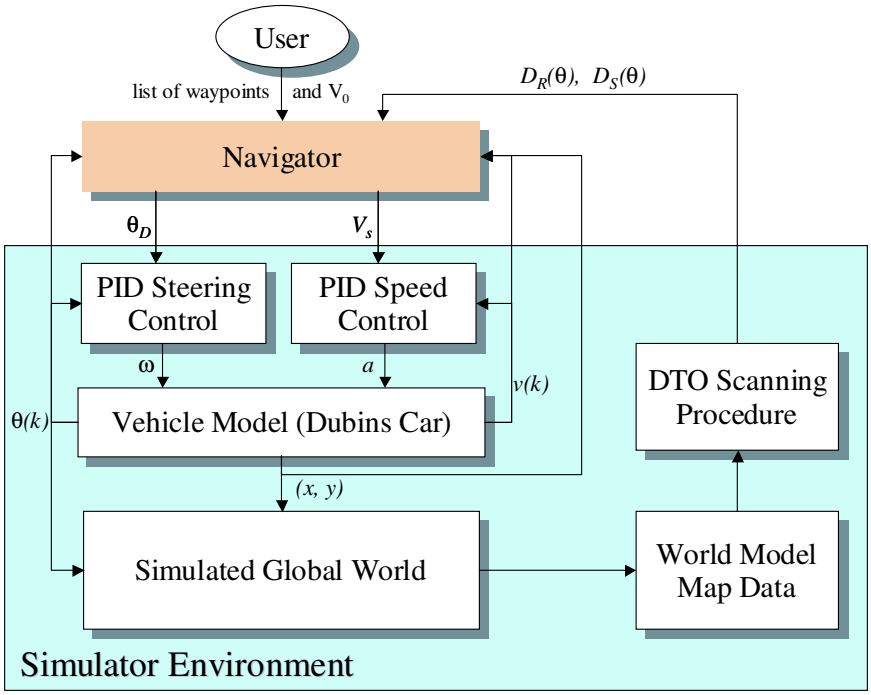


Figure 4.13: The simulator structure.

θ_D generated by the steering controller are displayed too. The vehicle's speed and orientation records are displayed in the views under the "global map view".

Figure 4.13 shows the simulator's structure, which implements a simple version of the generic architecture that we discussed in Chapter 2. Dubins' car model with a minimum turning radius is used as the vehicle model in this simulator. The discrete equations of the AGV are as follows:

$$\begin{cases} x(k+1) = x(k) + Tu(k)\cos\theta(k) \\ y(k+1) = y(k) + Tu(k)\sin\theta(k) \\ u(k+1) = u(k) + Ta(k) \\ \theta(k+1) = \theta(k) + T\omega(k) \end{cases} \tag{4.7}$$

where $x(k)$ and $y(k)$ are the position coordinates. $\theta(k)$ is the vehicle orientation. $u(k)$ stands for the linear velocity, which is assumed to be nonnegative. The acceleration $a(k)$ and the angular velocity ω are the control variables. T is the sampling time. The model is subject to the constraint:

$$\left|\frac{\omega}{u}\right| \leq \frac{1}{R} \tag{4.8}$$

so that a minimum turning radius R is imposed.

Both prior known and unknown obstacles are randomly generated over the terrain at the beginning of a simulation, and the prior known obstacles are displayed in the "global map view". User then selects a series of waypoints to connect the starting point and the final goal point. After that, a close-loop is established as shown in Figure 4.13 and the navigator guides the AGV to follow the waypoints one by one while avoiding obstacles. If the vehicle can reach the final goal point in reasonable time, then we say the experiment succeeds. Otherwise, if the vehicle gets stuck or cannot reach the final goal point for a long time, then we say the experiment fails. Figure 4.14 shows the AGV trajectory over such a terrain in an experiment. The trajectory of the AGV, the smooth (blue) line in the "global map view", indicates that the AGV follows the waypoints, avoids all obstacles and reaches the final goal point smoothly and safely.

The navigation algorithm has been tested in the simulator with different obstacle densities for more than 150 times and more than 90% of them succeeded[11]. The experiments failed in two kinds of situations. For one, there was no feasible paths

[11]The percentage of success experiments varies with the number and size of the obstacles in the terrain.

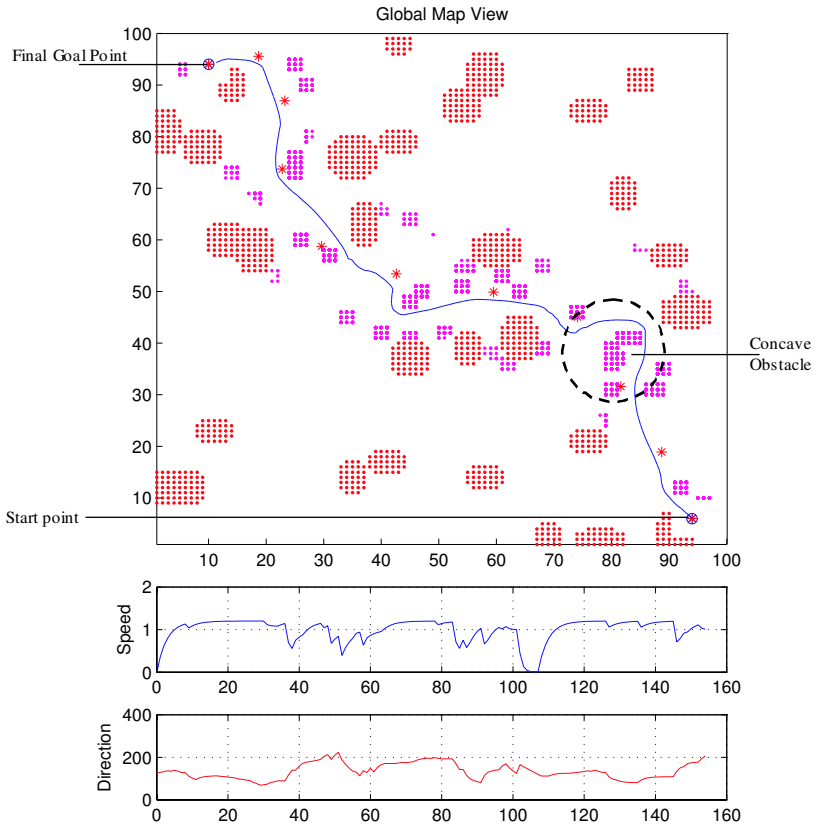


Figure 4.14: One experiment result.

go reach the goal point, which could happen since the obstacles are generated randomly. For another, the AGV was stuck because the waypoints misled the AGV into a wrong fork. The experiment results show that the above real-time fuzzy navigation algorithm is not sufficient in finding a path through very complicated environments. Nevertheless, it is sufficient for most normal applications, such as the DGC events. Also, to deal with the difficult situations, the navigator can request new goals and new waypoints from the higher level.

4.4.2 Tests on Small Robots

We have also tested the navigation algorithm on a Pioneer P3-AT rover equipped with one LIDAR and several SONARs as external environment sensors that provide accurate scanning DTO value directly to the navigator. The Pioneer P3-AT, developed by ActivMedia Robotics, drives by four reversible DC motors, each equipped with an optical quadrature shaft encoder for speed sensing and position-heading dead-reckoning.

Figure 4.15 shows a test on the small robot: the robot goes through many obstacles and gets to the other end of a hallway. We put an obstacle closely behind a gate formed by two obstacles, which is difficult for many navigation algorithms especially for the potential-field ones. In these tests, although the robot was surrounded by obstacles including the walls, our navigation algorithm was still able to find a path going along the corridor without touching any obstacles. We repeated the tests for over 20 times with different obstacles positions and configurations. Although there are cumulate errors in position-heading dead-reckoning, we only failed twice because of the extreme closeness of certain obstacles.

4.4.3 Implementation in ION

The performance of the navigator designed in this paper was demonstrated by ION in the DARPA GC05 event. ION completely traversed the NQE course successfully four times, which fully exhibited the obstacle-avoidance and goal-approaching capabilities of the navigation module and our AGV system.

Figure 4.16 shows ION successfully passing a car in the NQE. Figure 4.17 shows the path-planning result of ION in the two-car-passing section in the NQE. The

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

Figure 4.15: An experiment on indoor robot.

Figure 4.16: ION's passing a car in NQE, GC05.

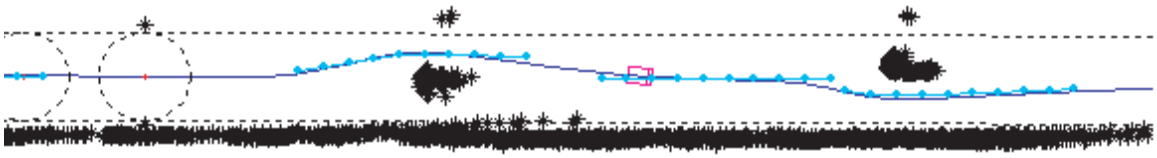


Figure 4.17: The example of ION's path planning in the two-car-passing section at NQE, GC05. The black stars are the obstacles recovered from ION's sensor logs. Two cars are placed in a corridor of 15ft half-width, as shown in the figure. The distance between the two cars is about 100ft. ION passes the two cars, from left to right in the figure, at the speed of 10mph, the highest speed permitted in the section.

dotted lines are the section boundary, the solid line is the trajectory that ION runs and the dark stars are the detected obstacles. Three sections of the 10-dot-curve are the planned path, which avoids the obstacles and stays within the section boundaries. Most planned paths are not shown in order to make the figure clear. ION's trajectory and planned paths almost overlap, which shows that our skill level controllers follows the planned path very well.

In the DGC05 final, ION covered about 30 miles in the Nevada Desert with complete autonomous operations, and finished as the 10th furthest.

4.5 Conclusion

This chapter presents a real-time AGV navigation system utilizing fuzzy logic. The navigation system distributes its task into two controllers, the steering controller and the speed controller. The steering fuzzy controller, by applying several fuzzy rules to a basic steering strategy, can lead the AGV to the goals smoothly and avoids the obstacles at the same time. The speed controller prevents the AGV from colliding with obstacles or rolling over. These two controllers work collaboratively.

Not only the simulation but also its physical implementations in both small robot and ION show the success of the fuzzy controllers and justify their performance.

CHAPTER 5

SMOOTH PATH PLANNING FOR CAR-LIKE AUTONOMOUS GROUND VEHICLES

5.1 Introduction

In this chapter, we will consider the problem of generating *smooth* paths in the navigation module. We call a path is *smooth* if the derivative of the unit tangent vector along the path is piece-wise continuous.

As described in the previous chapters, one of the skill level controllers' tasks is to keep the position and heading of the AGV continuously aligned with the paths that the navigation module generates. It is important for the navigation module to generate *smooth* paths, because:

- It is physically impossible for a car-like vehicle to change its heading without changing its position. Therefore, errors in position and heading will occur at each points where the the derivative of a path is not continuous. Proper feedforward and feedback control designs may be able to reduce the transition errors. However, the rapid error recovery requires rapid changes in steering which may not be permitted by the dynamic and kinematic constraints on the AGV.

- If the derivative of the tangent vector along the path is discontinuous at a point, then the vehicle has to stop and reorient its front wheel steering angle at the point, since the front wheel steering angle cannot be changed instantaneously. In this case, the path-following efficiency is sacrificed.

5.1.1 Preliminaries on Plane Curves

We first introduce some preliminaries on plane curves [89] that will be used in this chapter.

We describe a curve on $\mathbf{R}^2$ plane by parameterization: a curve is a continuous function from a subset of $\mathbf{R}$ into $\mathbf{R}^2$. Consider the parameterized curve $\mathbf{r}(t) = (x(t),\ y(t))$. Let $s(t)$ be the arc length from the point on the curvature at $t = t_0$ to the general point on the curve. Then

$$s(t) = \int_{t_0}^{t} ds = \int_{t_0}^{t} \sqrt{\dot{x}^2 + \dot{y}^2} dt = \int_{t_0}^{t} \|\dot{\mathbf{r}}(t)\| dt \tag{5.1}$$

where $\|\cdot\|$ is the Euclidean norm defined in $\mathbf{R}^2$. A parameterization $\mathbf{r}(t)$ is said to be *regular* if $\|\dot{\mathbf{r}}(t)\|$ is never zero for any t in its domain.

Let θ be the angle between the tangent of the curve at a point on the curve and the positive direction of the s-axis, shown in Figure 5.1. Naturally, the curvature k is defined to be the rate of curving as a point moves along the curve, i.e., $k = d\theta/ds$.

Lemma 5.1: *If the curve* $\mathbf{r}(t) = [x(t)\ y(t)]^T$ *is second-order differentiable,then its curvature is given by*

$$k(t) = \frac{\dot{x}(t)\ddot{y}(t) - \ddot{x}(t)\dot{y}(t)}{(\dot{x}(t)^2 + \dot{y}(t)^2)^{3/2}} \tag{5.2}$$

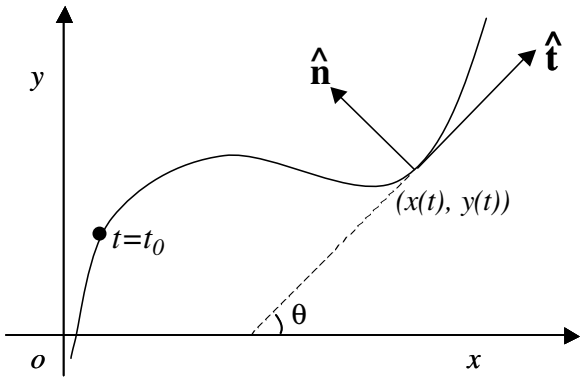


Figure 5.1: The Frenet frame at a point on the curve.

The Frenet frame, a special set of orthogonal axes or frame, are given by two orthogonal vectors $\hat{\mathbf{t}}$ and $\hat{\mathbf{n}}$, where $\hat{\mathbf{t}} = \dot{\mathbf{r}}(t)/\|\dot{\mathbf{r}}(t)\|$ is the unit tangent vector. As shown in Figure 5.1, $\hat{\mathbf{t}} = (\cos\theta,\ \sin\theta)$ and $\hat{\mathbf{n}}$ is a 90° counterclockwise rotation of $\hat{\mathbf{t}}$, so that $\hat{\mathbf{n}} = (-\sin\theta,\ \cos\theta)$. It follows that

$$\begin{bmatrix} \dot{\hat{\mathbf{t}}} \\ \dot{\hat{\mathbf{n}}} \end{bmatrix} = \begin{bmatrix} 0 & kv \\ -kv & 0 \end{bmatrix} \begin{bmatrix} \hat{\mathbf{t}} \\ \hat{\mathbf{n}} \end{bmatrix} \tag{5.3}$$

where $v = ds/dt = \|\dot{\mathbf{r}}(t)\|$.

We adopt the following definitions on the continuity of plane curves from [66].

Definition 5.1: *(G^1- and G^2-Curves)* Consider the parameterized curve $\mathbf{r}(t) = (x(t),\ y(t))$. We say $\mathbf{r}(t)$ is a G^1-curve, if it has first-order geometric continuity: $\mathbf{r}(t)$ is *regular* and $\hat{\mathbf{t}}$ is continuous along the the curve. We say $\mathbf{r}(t)$ is a G^2-curve, if it has second-order geometric continuity: $\mathbf{r}(t)$ is a G^1-curve and its curvature k is continuous along the the curve. The image of a G^2-Curve is called a *smooth path*.

5.1.2 Smooth Path-Planning Problem Setup

As we have described in Section 1.2, there are many path-planning algorithms that generate non-smooth paths by means of generating a sequence of waypoints that connects the initial position to the goal position without crossing obstacles, such as the visibility graph methods, the Voronoi diagrams, the cell decomposition methods and the A* algorithm, etc. Also, in some applications, such as the DGCs, waypoints are provided and the AGV is required to follow the waypoints.

Therefore, we concern the problem of connecting the waypoints by a *smooth* path, i.e. a G^2-curve.

Problem 5.1 The G^2-interpolation problem: *Given a series of N separated waypoints $\mathbf{P} := \{P_i = (x_i,\ y_i)^T | i = 1, 2, \cdots N\} \subset \mathbf{R}^2$, find a curve $\mathbf{r} : [a,\ b] \to \mathbf{R}^2$ such that:*

- $\mathbf{r}$ *is a G^2-curve;*
- $\exists a \leq t_1 < t_2 < \cdots < t_N \leq b$ *s.t.* $\mathbf{r}(t_i) = P_i = (x_i,\ y_i)^T,\ i = 1, 2, \cdots N.$

For convenience, we usually choose the parameter t to lie in the interval $[0,\ 1]$. It is also possible that the curve may be generated by directly in its Cartesian form

$$y = f(x; \mathbf{P}). \tag{5.4}$$

Solutions to the G^2-interpolation problem is important because of the following property proved by Piazzi et al in [66, 62, 90]:

Property 5.1 *A curve on $\mathbf{R}^2$ plane is generated by a Dubins car [58] with a* continuous *steering control input, if and only if the path is a G^2-curve.*

For N separated waypoints, there is a unique polynomial of degree N in form of

$$y = a_{N-1}x^{N-1} + \cdots + a_1 x + a_0 \tag{5.5}$$

where the coefficients $a_i,\ i = 1, 2, \cdots, N$ are to be determined so that the curve passes through all of the points. Lagrange polynomials and Hermite polynomials are commonly used for polynomial interpolation. The latter is useful when we have not only the values of (x_i, y_i) but also the values of the gradient $y'_i = (dy/dx)_i$. The polynomials, which is differentiable up to any degrees, can solve the G^2-interpolation problem. However, there are two major drawbacks. First, when N is a large number (for example, $N \simeq 3000$ in DGC2), it is not sensible to use one polynomial to represent a curve over a large range of values, because the high degree polynomial tends to introduce unrepresentative oscillations [89]. Second, the above polynomial curve might not passing the waypoints in the order they are described.

We can overcome the problems by using low-degree polynomials in a piecewise manner. The piecewise interpolation problem is defined as follows.

Problem 5.2 The piecewise G^2-interpolation problem: *Given a series of separate waypoints* $\mathbf{P} := \{P_i = (x_i,\ y_i)^T | i = 1, 2, \cdots N\} \subset \mathbf{R}^2$, *find a series of continuous mapping* $\mathbf{r}_i : [0, 1] \to \mathbf{R}^2, i = 1, 2, \cdots, N-1$ *be the curves connecting the points in* $\mathbf{P}$, *such that:*

(c1) each $\mathbf{r}_i(t)$ *is a* G^2*-curve;*

(c2) $\mathbf{r}_i(0) = P_i$, *i.e.* $x_i(0) = x_i, \quad y_i(0) = y_i$;

(c3) $\mathbf{r}_i(1) = P_{i+1} = \mathbf{r}_{i+1}(0)$, *i.e.* $x_i(1) = x_{i+1}, \quad y_i(1) = y_{i+1}$;

(c4) $\hat{\mathbf{t}}_i(1) = \hat{\mathbf{t}}_{i+1}(0)$;

(c5) $\dot{\hat{\mathbf{t}}}_i(1) = \dot{\hat{\mathbf{t}}}_{i+1}(0)$*, which is equivalent to* $k_i(1) = k_{i+1}(0)$*.*

where $\mathbf{r}_i(t) = (x_i(t),\ y_i(t))^T$*,* $t \in [0, 1]$*, is the* i*th curve connecting waypoints* P_i *and* P_{i+1}*, and* $\hat{\mathbf{t}}_i$ *is the unit tangent vector in the Frenet frame along the curves.*

Property 5.2 *The solutions to the piecewise* G^2*-interpolation problem form a* G^2*-curve that solves the* G^2*-interpolation problem (Problem 5.1).*

Proof: Define a curve $\mathbf{r} : [0, N-1] \to \mathbf{R}^2$ by $\mathbf{r}(t) = \mathbf{r}_i(t - i + 1)$, for some $i \in \{1, 2, \cdots, N-1\}$ such that $t \in (i-1, i]$. Also, let $\mathbf{r}(0) = \mathbf{r}_1(0)$.

Then the curve $\mathbf{r}$ is second-order differentiable and the curvature along curve $\mathbf{r}$ is continuous, i.e., it is a G^2-curve. Also, let $t_i = i - 1,\ i = 1, 2, \cdots, N$, so that $0 = t_1 < t_2 < \cdots < t_N = N - 1$. Then $\mathbf{r}(t_i) = \mathbf{r}_{i-1}(1) = P_i$, for $i > 1$ and $\mathbf{r}(t_1) = P_1$.

Therefore, the curve $\mathbf{r}$ is a solution to Problem 5.1. ■

5.2 Smooth Path Planning

5.2.1 Polynomial Piecewise G^2-Interpolation

We can use low-degree polynomials to solve the piecewise G^2-interpolation problem. The resulting curve is called a *spline*.

According to the theory of planar curves [89], cubic polynomials (natural cubic splines or cubic B-splines) are widely used to solve the piecewise G^2-interpolation problems, such as [63] and [61]. To find such a cubic spline, however, one has to use all N position vectors defined in $\mathbf{P}$ to calculate $O(N)$ parameters. One significant disadvantage is that local modifications of one point in $\mathbf{P}$ require that the whole spline curves be recomputed.

In [66, 62, 90], the authors proposed to use a quintic spline to interpolate two points with given direction and curvature at each points as boundary conditions.

In the following, we shall consider using only local position information to generate a quartic spline that solves Problem 5.2.

5.2.2 Quartic Piecewise G^2-Interpolation

Let $\mathbf{r}_i$ be the ith quartic polynomial curve connecting waypoints P_i and P_{i+1}. We shall find such a polynomial curve only using the position information of the four waypoints: $\{P_{i-1},\ P_i,\ P_{i+1},\ P_{i+2}\}$.

First, we define four quartic functions as

$$f_i(t) = a_{i4}t^4 + a_{i3}t^3 + a_{i2}t^2 + a_{i1}t + a_{i0}, \quad i = 1, 2, 3, 4, \quad \text{and } t \in [0,\ 1] \tag{5.6}$$

where $a_{ij},\ (i = 1, 2, 3, 4,\ j = 0, 1, 2, 3, 4)$, are the parameters we shall determine. And we define the quartic interpolation curve in the following form

$$\begin{aligned} \mathbf{r}_i(t) = \begin{bmatrix} x_i(t) \\ y_i(t) \end{bmatrix} &= [P_{i-1},\ P_i,\ P_{i+1},\ P_{i+2}] \begin{bmatrix} f_1(t) \\ f_2(t) \\ f_3(t) \\ f_4(t) \end{bmatrix} \\ &= M_i A \begin{bmatrix} t^4 & t^3 & t^2 & t & 1 \end{bmatrix}^T \end{aligned} \tag{5.7}$$

where $M_i := [P_{i-1},\ P_i,\ P_{i+1},\ P_{i+2}]$, and the A is the parameter matrix that needs to be computed,

$$A := \begin{bmatrix} a_{14} & a_{13} & a_{12} & a_{11} & a_{10} \\ a_{24} & a_{23} & a_{22} & a_{21} & a_{20} \\ a_{34} & a_{33} & a_{32} & a_{31} & a_{30} \\ a_{44} & a_{43} & a_{42} & a_{41} & a_{40} \end{bmatrix}.$$

It is obvious that the quartic polynomials $\mathbf{r}_i(t)$ defined in (5.7) are G^2-curves since they are second-order differentiable. To satisfy the conditions (c2) and (c3) in Problem

5.2, we have following equations for arbitrarily selected waypoints $\{P_{i-1},\ P_i, \cdots, P_{i+3}\}$:

$$\begin{aligned} \mathbf{r}_i(0) = P_i \quad &\Rightarrow \quad [P_{i-1},\ P_i,\ P_{i+1},\ P_{i+2}]A[0\ 0\ 0\ 0\ 1]^T = P_i \\ &\Rightarrow \quad A[0\ 0\ 0\ 0\ 1]^T = [0\ 1\ 0\ 0]^T \end{aligned} \tag{5.8}$$

and

$$\begin{aligned} \mathbf{r}_i(1) = P_{i+1} \quad &\Rightarrow \quad [P_{i-1},\ P_i,\ P_{i+1},\ P_{i+2}]A[1\ 1\ 1\ 1\ 1]^T = P_{i+1} \\ &\Rightarrow \quad A[1\ 1\ 1\ 1\ 1]^T = [0\ 0\ 1\ 0]^T \end{aligned} \tag{5.9}$$

Also, condition (c4) in Problem 5.2 requires that $\hat{\mathbf{t}}_i(1) = \hat{\mathbf{t}}_{i+1}(0)$, where $\hat{\mathbf{t}}_i(t) = \dot{\mathbf{r}}_i(t)/\|\dot{\mathbf{r}}_i(t)\|$. Since $\dot{\mathbf{r}}_i(t) = [\dot{x}_i(t), \dot{y}_i(t)]^T = M_i A[4t^3\ 3t^2\ 2t\ 1\ 0]^T$, it implies that

$$\frac{\dot{y}_i(1)}{\dot{x}_i(1)} = \frac{\dot{y}_{i+1}(0)}{\dot{x}_{i+1}(0)} \quad \forall\{P_{i-1},\ P_i,\ P_{i+1},\ P_{i+2},\ P_{i+3}\}$$

Since the waypoints $\{P_{i-1}, \cdots, P_{i+3}\}$ can be selected arbitrarily, it implies that

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} A \begin{bmatrix} 4 \\ 3 \\ 2 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} A \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \tag{5.10}$$

Condition (c5) in Problem 5.2 requires that $k_i(1) = k_{i+1}(0)$. Since $\ddot{\mathbf{r}}_i(t) = M_i A[12t^2\ 6t\ 2\ 0\ 0]^T$ and that the waypoints $\{P_{i-1}, \cdots, P_{i+3}\}$ can be selected arbitrarily, by applying Lemma 5.1, we can concluding that

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} A \begin{bmatrix} 12 \\ 6 \\ 2 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} A \begin{bmatrix} 0 \\ 0 \\ 2 \\ 0 \\ 0 \end{bmatrix} \tag{5.11}$$

Therefore, the parameter matrix A, which is a 4×5 matrix, has to satisfy (5.8), (5.9), (5.10) and (5.11), which provide 4, 4, 5, and 5 independent equations, respectively. Therefore, we have two freedom in selecting the values in matrix A.

By solving the 18 equations specified in (5.8), (5.9), (5.10) and (5.11), we obtain

$$A = \begin{bmatrix} \alpha & -3\alpha & 3\alpha & -\alpha & 0 \\ 4\alpha+\beta-3 & -8\alpha-3\beta+8 & 3\alpha+3\beta-6 & \alpha-\beta & 1 \\ \alpha+4\beta-3 & 4-\alpha-8\beta & 3\beta & \beta & 0 \\ \beta & -\beta & 0 & 0 & 0 \end{bmatrix} \tag{5.12}$$

where α and β are arbitrary non-zero real values. We pack these values together to form a parameter vector over $\mathbf{R}^2$ as $\mu := [\alpha\ \beta]^T$, so that the curve $\mathbf{r}_i$ can be denoted as $\mathbf{r}(t;\mu;M_i)$. Denote

$$F(t;\mu) := \begin{bmatrix} f_1(t;\mu) \\ f_2(t;\mu) \\ f_3(t;\mu) \\ f_4(t;\mu) \end{bmatrix} = A(\mu) \begin{bmatrix} t^4 \\ t^3 \\ t^2 \\ t \\ 1 \end{bmatrix} \tag{5.13}$$

then

$$\mathbf{r}(t;\mu;M_i) = M_i F(t;\mu) \tag{5.14}$$

where

$$\begin{cases} f_1(t;\mu) = \alpha(t-1)^3 t \\ f_2(t;\mu) = (t-1)\left[\alpha(4t^2-4t-1)t + \beta(t-1)^2 t - (t-1)^2(3t+1)\right] \\ f_3(t;\mu) = t\left[\alpha(t-1)t^2 + \beta(t-1)(4t^2-4t-1) + t^2(4-3t)\right] \\ f_4(t;\mu) = \beta(t-1)t^3 \end{cases} \tag{5.15}$$

Then, given any series of separate waypoints $\mathbf{P} := \{P_i = (x_i,\ y_i)^T | i = 1, 2, \cdots N\} \subset \mathbf{R}^2$, the parametric curves $\mathbf{r}(t;\mu;M_i) = M_i F(t,\mu)$ satisfy the piecewise G^2-interpolation conditions (c1)-(c5) in Problem 5.2 for all $\mu \in \mathbf{R}^2 \setminus \{\alpha\beta = 0\}$.

5.2.3 Properties of the Quartic Splines

We shall investigate several quartic spline properties that are related to the parameter μ.

Definition 5.1 *We call two series of waypoints $W = \{w_i\} \subset \mathbf{R}^2$ and $V = \{v_i\} \subset \mathbf{R}^2$ are* **linear equivalent**, *if there exist nonsingular $T \in \mathbf{R}^{2\times 2}$ and $B \in \mathbf{R}^2$, such that $v_i = Tw_i + B$ for any $i = 1, 2, ..., N$. Short handed as $V = TW + B$.*

Theorem 5.1: **Linear Transformation Theorem:** *Given any nonsingular* $T \in \mathbf{R}^{2\times 2}$*, non-trivial* $B \in \mathbf{R}^2$ *and two linear equivalent series of waypoints* $W = \{w_i\} \subset \mathbf{R}^2$ *and* $V = \{v_i\} \subset \mathbf{R}^2$ *with* $V = TW + B$*, then the quartic* G^2*-interpolation curves* $\mathbf{r}(t; [\alpha, \beta]^T; V) = T\mathbf{r}(t; [\alpha, \beta]^T; W) + B,\ \forall t \in [0, 1]$*, if and only if* $\alpha + \beta = 1$*.*

Proof: Let $\mu = [\alpha, \beta]^T$. By Definition 5.1 and (5.13):

$$\begin{aligned}
\mathbf{r}(t; \mu; V) &= [v_{i-1}, v_i, v_{i+1}, v_{i+2}]F(t; \mu) \\
&= \left(T[w_{i-1}, w_i, w_{i+1}, w_{i+2}] + [B\ B\ B\ B]\right) F(t; \mu) \\
&= T\mathbf{r}(t; \mu; W) + B[1\ 1\ 1\ 1]F(t; \mu) \\
&= T\mathbf{r}(t; \mu; W) + B\left(f_1(t; \mu) + f_2(t; \mu) + f_3(t; \mu) + f_4(t; \mu)\right) \\
&= T\mathbf{r}(t; \mu; W) + B\left(6(\alpha + \beta - 1)t^2(t-1)^2 + 1\right)
\end{aligned}$$

Sufficiency: when $\alpha + \beta = 1$,

$$\begin{aligned}
\mathbf{r}(t; \mu; V) &= T\mathbf{r}(t; \mu; W) + B\left(6(\alpha + \beta - 1)t^2(t-1)^2 + 1\right) \\
&= T\mathbf{r}(t; \mu; W) + B.
\end{aligned}$$

Necessity: if $\mathbf{r}(t; [\alpha, \beta]^T; V) = T\mathbf{r}(t; [\alpha, \beta]^T; W) + B,\ \forall t \in [0, 1]$, since B is non-trivial, then $(\alpha + \beta - 1)t^2(t-1)^2 = 0,\ \forall t \in [0, 1]$ therefore $\alpha + \beta = 1$. ■

Corollary 5.1: *Given any four waypoints* $M = \{P_0,\ P_1,\ P_2,\ P_3\} \subset \mathbf{R}^2$ *that stay on a line in order, i.e.* $P_i = (x_i,\ y_i)^T,\ i = 0, 1, 2, 3$ *and* $ax_i + by_i = c$ *for some* a, b, c *that* $a \neq 0$ *or* $b \neq 0$*. Let* $\mathbf{r}(t; \mu; M)$ *be the quartic polynomial interpolation curve that connects waypoints* P_1 *and* P_2 *defined in (5.13). If* $\alpha + \beta = 1$*, then the path generated by* $\mathbf{r}(t; \mu; M)$ *is actually the line segment that connects* P_1 *and* P_2*, i.e.*

$$[a\ b]\mathbf{r}(t; \mu; M) = c, \quad \forall t \in [0,\ 1].$$

Proof: Let $B = \begin{bmatrix} 0 \\ -c \end{bmatrix}$ and find any number $d \in \mathbf{R}$ such that $T = \begin{bmatrix} 1 & d \\ a & b \end{bmatrix}$ is a nonsingular matrix, and let $N = TM + B = \{\bar{P}_i | \bar{P}_i = [\bar{x}_i \ \bar{y}_i]^T = TP_i + B, \ i = 0, 1, 2, 3\}$. Then $\bar{y}_i = [a \ b]P_i - c = 0$, therefore,

$$[0 \ 1]\mathbf{r}(t; \mu; N) = [0 \ 1]NF(t; \mu) = [0 \ 0 \ 0 \ 0]F(t; \mu) = 0, \quad \forall t \in [0, 1].$$

Also, when $\alpha + \beta = 1$, by the linear transformation Theorem 5.1, we have $\mathbf{r}(t; \mu; TM + B) = T\mathbf{r}(t; \mu; M) + B$ so that $[0 \ 1](T\mathbf{r}(t; \mu; M) + B) = 0$. Hence,

$$[a \ b]\mathbf{r}(t; \mu; M) = c$$

■

Theorem 5.2: **Symmetric Theorem:** *Let* $W = \{P_1, P_2, P_3, P_4\}$ *and* $V = \{P_4, P_3, P_2, P_1\}$ *are any two series of waypoints that are in reverse orders. Let* $\mu = [\alpha, \beta]^T$. *Then, the quartic interpolation curves* $\mathbf{r}(t; \mu; W)$ *and* $\mathbf{r}(t; \mu; V)$ *overlap, if and only if* $\alpha = \beta$. *Moreover, if* $\alpha = \beta$, $\mathbf{r}(t; \mu; W) = \mathbf{r}(1 - t; \mu; V)$.

Proof: Sufficiency: if $\alpha = \beta$, then by (5.15):

$$\begin{cases} f_1(1 - t; \mu) = f_4(t; \mu) \\ f_2(1 - t; \mu) = f_3(t; \mu) \end{cases}$$

So, by the theorem conditions, it is clear that

$$\begin{aligned} \mathbf{r}(1 - t; \mu; V) &= VF(1 - t; \mu) \\ &= P_4 f_1(1 - t; \mu) + P_3 f_2(1 - t; \mu) + P_2 f_3(1 - t; \mu) + P_1 f_4(1 - t; \mu) \\ &= P_4 f_4(t; \mu) + P_3 f_3(t; \mu) + P_2 f_2(t; \mu) + P_1 f_1(t; \mu) \\ &= WF(t; \mu) \\ &= \mathbf{r}(t; \mu; W) \end{aligned}$$

Therefore, the two curves overlap.

Necessity: Since the waypoints in W and V can be arbitrary selected, we choose $W = [P_1\ P_2\ P_3\ P_4] = \begin{bmatrix} 0 & x_2 & x_3 & x_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$, and V is the reverse of W. Therefore,

$$\mathbf{r}(t;\mu;W) = \begin{bmatrix} * \\ f_4(t;\mu) \end{bmatrix} = \begin{bmatrix} * \\ \beta(t-1)t^3 \end{bmatrix}$$

and

$$\mathbf{r}(t;\mu;V) = \begin{bmatrix} * \\ f_1(t;\mu) \end{bmatrix} = \begin{bmatrix} * \\ \alpha(t-1)^3 t \end{bmatrix}$$

Since $\mathbf{r}(t;\mu;W)$ and $\mathbf{r}(t;\mu;V)$ overlap, we have $\forall t \in [0,1], \exists\ \bar{t} \in [0,1]$ such that $\mathbf{r}(t;\mu;W) = \mathbf{r}(\bar{t};\mu;V)$.

Hence,

$$\beta(t-1)t^3 = \alpha(\bar{t}-1)^3\bar{t}. \tag{5.16}$$

Since equation (5.16) is true for all $t, \bar{t} \in [0,1]$, so the maximum of the left side should equal the maximum of the right side, which implies $\alpha = \beta$.

■

5.2.4 Quartic Splines Examples

According to the Theorem 5.1 and Theorem 5.2, parameter $\mu = [0.5, 0.5]^T$ preserves the invariance under linear transformation and symmetry under waypoints reversion. We shall use this value in the following examples.

Lane Change Case

We use $\mathbf{P} = \begin{bmatrix} 0 & 5 & 10 & 15 & 20 & 25 & 35 \\ 0 & 0 & 0 & 5 & 10 & 10 & 10 \end{bmatrix}$ to specify the series of waypoints for lateral lane change. Figure 5.2 compares the quartic piecewise G^2-curve to the commonly used cubic B-spline. Note that the cubic B-spline is generated in a "global"

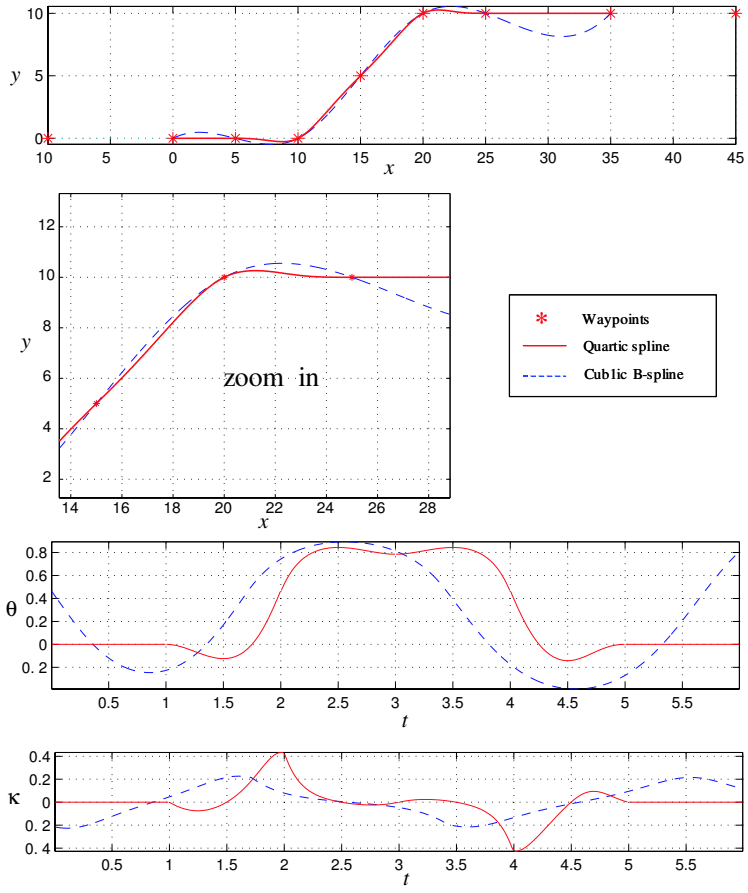


Figure 5.2: The quartic piecewise G^2-curve is compared to the cubic B-spline. The heading θ and curvature κ are compared too.

fashion by using all data in $\mathbf{P}$ at once, while the quartic spline is generated "locally" by using only four waypoints for each section. Two artificial waypoints are added in the quartic case, $\begin{bmatrix} -10 \\ 0 \end{bmatrix}$ is added to the beginning of $\mathbf{P}$ and $\begin{bmatrix} 45 \\ 10 \end{bmatrix}$ is appended to the end of $\mathbf{P}$. The example shows that quartic splines are better than the cubic B-spline in the sense of minimizing the offset from the line segments.

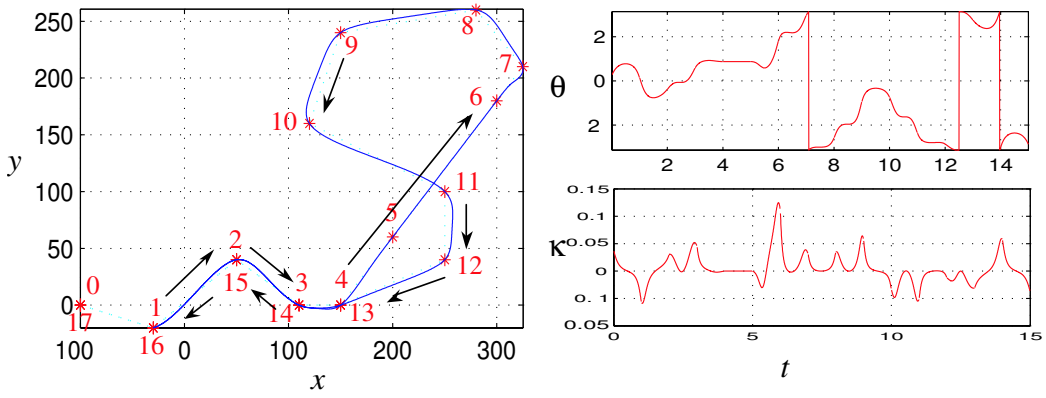


Figure 5.3: An example that shows two special properties of quartic spline interpolation: (1) Points 4, 5, 6 and 7 are in a line, so that interpolation between point 5 and 6 is actually a line segment; (2) Points 0-4 and 13-17 overlap in reverse order, so that the interpolation between (1,2) and (2,3) overlap with (15,16) and (14,15), respectively.

Line Segment and Symmetric Reversion Cases

Figure 5.3 shows two special properties of the quartic spline ($\alpha = \beta = 0.5$). First, when four points are in a line, then the quartic spline is reduced to a line segment, as proved in Corollary 5.1. Second, as shown in Theorem 5.2, the quartic splines overlap if only the order of the waypoints is reversed.

5.3 Optimal Quartic Splines

In this section, we shall concern the optimal quartic piecewise G^2-interpolation problem in the sense of minimum offset[12] from the line segments that connect the given waypoints because, in many cases, large offset from the line segments means high probability in meeting obstacles. For example, in DGC04 and DGC05, corridors

[12]People have used different criteria to evaluate a planned path, such as the curvature and the derivative of curvature along the path [63, 65].

of certain width were specified by the a series of waypoints and vehicles had to stay in the corridors, or the vehicle maybe disqualified.

Let $\overline{P_1P_2}$ represents the line segment that connects P_1 and P_2. Define the Euclidian distance from P to the line segment as follows:

$$D(P, \overline{P_1P_2}) := \min_{S \in \overline{P_1P_2}} \|P - S\|_2.$$

Problem 5.3 The Minimum offset quartic G^2-interpolation problem: *Given a series of separated waypoints* $\mathbf{P} := \{P_i = (x_i,\ y_i)^T | i = 1, 2, \cdots N\} \subset \mathbf{R}^2$. *Let* $\mathbf{r}_i$, $i = 1, 2, \cdots, N-1$, *be a quartic polynomial solution to the piecewise G^2-interpolation problem (Problem 5.2). Each* $\mathbf{r}_i$ *connects waypoint* P_i *and* P_{i+1}. *The optimal problem is to find the* $\mathbf{r}_i$*'s that minimize the offset of the line segments:*

$$\min_{\{\mathbf{r}_i\}} \ J := \max_i \max_{P \in \mathbf{r}_i} D(P, \overline{P_iP_{i+1}}) \tag{5.17}$$

Furthermore, the curves $\mathbf{r}_i$ *are subject to an upper bounded curvature, i.e.:*

$$|k_P| < k_{max} < \infty,\ \forall\ P \in \mathbf{r}_i \tag{5.18}$$

Since we have fixed $\alpha = \beta = 0.5$, Problem 5.3 seems trivial because only one quartic spline can be generated from the waypoint set $\mathbf{P}$. However, we can insert two (or more) waypoints, say Q_{i1} and Q_{i2}, into each line segment $\overline{P_iP_{i+1}}$, as shown in Figure 5.4. By doing so, each quartic curve $\mathbf{r}(t; \mu; [P_{i-1}\ P_i\ P_{i+1}\ P_{i+2}])$ is substituted by three (or more) quartic curves that connects $[P_i\ Q_{i1}]$, $[Q_{i1}\ Q_{i2}]$ and $[Q_{i2}\ P_{i+1}]$.

According to Corollary 5.1, since $Q_ij \in \overline{P_iP_{i+1}}, j = 1, 2$, we can conclude that $\mathbf{r}(t; \mu; [P_i\ Q_{i1}\ Q_{i2}\ P_{i+1}]) \subset \overline{P_iP_{i+1}}$. Therefore, there is no offset in the interpolation between Q_{i1} and Q_{i2}.

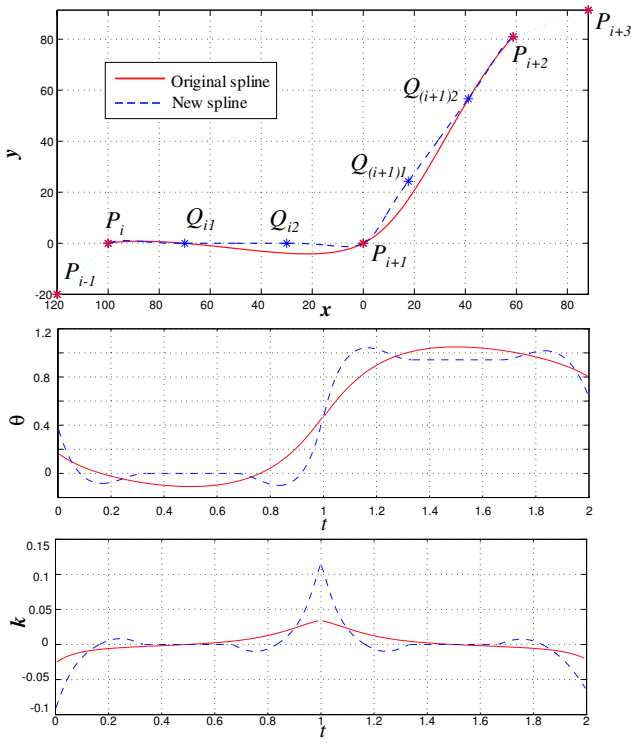


Figure 5.4: Example of quartic spline on inserted waypoints.

Also, consider two waypoint sets M_i and $\bar{M}_i$ that are similar up to scaling, i.e. $\bar{M}_i = \lambda M_i$ for some $\lambda \in \mathbf{R}$. Let $\mathbf{r}_i(t) = M_i F(t)$ and $\bar{\mathbf{r}}_i(t) = \bar{M}_i F(t)$, then, by Theorem 5.1, $\bar{\mathbf{r}}_i(t) = \lambda \mathbf{r}_i(t)$ so that

$$\max_{P \in \bar{\mathbf{r}}} D(P, \overline{(\lambda P_i)(\lambda P_{i+1})}) = \lambda \max_{P \in \mathbf{r}} D(P, \overline{P_i P_{i+1}})$$

So, the offset of the quartic curve is proportional to the distances between the waypoints. Therefore, inserting additional waypoints can reduce the offset cost defined in (5.17). Figure 5.4 shows an example of two quartic splines, one is generated on the original waypoint set and the other is generated on the waypoint set with inserted points.

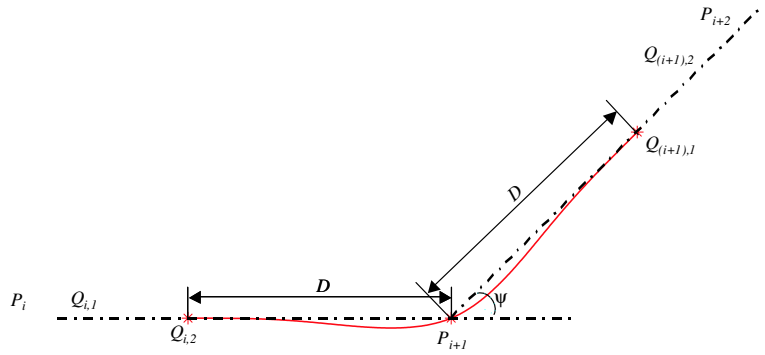


Figure 5.5: Example of inserting two waypoints, the $Q_{i,2}$ and $Q_{(i+1),1}$, around P_{i+1}.

On the other hand, inserting waypoints will cause the increase in the maximum curvature. Therefore, because of the maximum curvature constraint (5.18), we cannot insert waypoints arbitrary. In the following, we shall find the optimal waypoint insertion that satisfies the curvature constraint.

Without losing generality, we assume $P_i = [-L,\ 0]^T,\ P_{i+1} = [0,\ 0]^T$ are on the x-axis and P_{i+2} is off x-axis. Let $D = \|P_{i+1} - Q_{i2}\|$ be the distance from the inserted waypoints Q_{i2} to waypoint P_{i+1}, and let ψ be the direction change at waypoint P_{i+1}, as illustrated in Figure 5.5. Because reversing the order of the waypoints won't change the optimal result, we can conclude that $D = \|Q_{(i+1),1} - P_{i+1}\|$ too. Then, the quartic spline connecting Q_{i2} and P_{i+1} is determined by following four waypoints: $M := \begin{bmatrix} -D_1 & -D & 0 & D\cos\psi \\ 0 & 0 & 0 & D\sin\psi \end{bmatrix}$, where $D_1 > D$ and will be cancelled in the following calculations. Therefore, both the maximum offset and maximum curvature in the spline is determined by the values of D and ψ.

First, let $\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \mathbf{r}(t; M) = MF(t)$, then

$$y(t) = D\sin\psi * f_4(t) = \frac{1}{2}D\sin\psi(t-1)t^3,$$

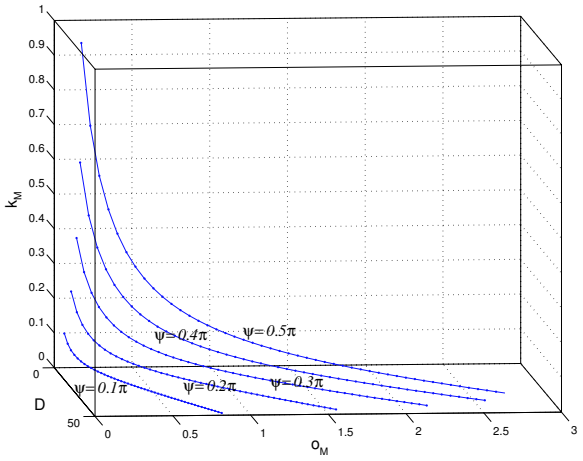


Figure 5.6: The relationship among D, the maximum offset cost o_M, and the maximum curvature k_M.

therefore, the maximum offset, denoted as o_M, can be obtained in the following:

$$o_M = \max_{t\in[0,\ 1]} |y(t)| = \frac{27}{512} D \sin\psi \tag{5.19}$$

Second, let k_M be the maximum curvature along the curve $\mathbf{r}(t; M)$. Let $e_0(t) = [t^4\ t^3\ t^2\ t\ 1]^T$, $e_1(t) = \dot{e}_0(t) = [4t^3\ 3t^2\ 2t\ 1\ 0]^T$, and $e_2(t) = \ddot{e}_0(t) = [12t^2\ 6t\ 2\ 0\ 0]^T$. Then

$$\begin{cases} \dot{x}(t) = [-D_1\ -D\ 0\ D\cos\psi] A e_1(t) \\ \ddot{x}(t) = [-D_1\ -D\ 0\ D\cos\psi] A e_1(t) A e_1(t) \\ \dot{y}(t) = [0\ 0\ 0\ D\sin\psi] A e_1(t) \\ \ddot{y}(t) = [0\ 0\ 0\ D\sin\psi] A e_1(t) \end{cases}$$

And, according to Lemma 5.1, the curvature is given by

$$k(t) = \frac{\dot{x}(t)\ddot{y}(t) - \ddot{x}(t)\dot{y}(t)}{(\dot{x}(t)^2 + \dot{y}(t)^2)^{3/2}}$$

After tedious calculation, we obtain that

$$k_M = \max_{t\in[0,\ 1]} |k(t)| = \frac{12\sin\frac{\psi}{2}}{D(1+\cos\psi)} \tag{5.20}$$

Therefore, given any curvature constraint, we can find the minimum distance for inserting waypoints and calculate the maximum offset according to equation (5.19) and (5.20). Figure 5.6 shows the relationship among D, o_M, and k_M.

5.4 Concluding Remarks

In this chapter, we discuss the problem of generating a smooth path that passes through a set of waypoints.

To generate the smooth path, we introduce the piecewise interpolation method instead of adopting "global" interpolation methods. Since the "global" methods may require heavy computations on all information of the waypoints and probably result in unrepresentative oscillations. In contrast, our piecewise interpolation methods only requires the position information of nearby waypoints. Quartic polynomials are used to solve the piecewise interpolation problem and the polynomial coefficient matrix is computed from boundary conditions that are specified in equations (5.8), (5.9), (5.10) and (5.11). In order to make sure that the distance between the line segments of the given waypoints and the smooth path is small so that the vehicle stays right on the path, we formulate and study the minimum offset smooth interpolation problem. With the given way points and maximum turning curvature, the minimum offset can be calculated by using quartic splines.

CHAPTER 6

MANEUVER AUTOMATION FOR CAR-LIKE GROUND VEHICLES

6.1 Introduction

The smooth path-planning algorithm discussed in the previous chapter provides us a method to generate smooth paths for a car-like vehicle to follow. Nevertheless, in some cases, it is impossible to generate a smooth path for the vehicle, either because the maximum turning curvature of the vehicle can not be satisfied or because the maximum offset of the generated smooth path is too large. For example, park the vehicle in a parking garage or drive the vehicle in a narrow and sharp-turn situation. Therefore, in these cases, the vehicle needs to maneuver to a new configuration from where the following path is smooth and the curvature is within the vehicle's limit. Generally, the maneuver is accomplished by performing a series of back-and-forth operations, called *maneuver operations*.

In this chapter, we shall consider the problem of driving such a car-like vehicle from one configuration (position and orientation) to another configuration. The problem is also called the maneuver planning problem, or the maneuver automation problem.

The maneuver automation problem is difficult in general [23, 55, 91]. And it becomes more difficult to solve the problem amidst obstacles.

6.2 Simplified Car-Like Vehicle Model

6.2.1 Kinematic Model

In the maneuver automation problem, because the vehicle speed is very low in general, so we can use simplified vehicle model that ignores the vehicle dynamics.

Let the state of a car-like vehicle be represented by a triple $\xi = (x, y, \theta) \in \mathcal{C} = \mathbf{R}^2 \times S$, where $S := [0,\ 2\pi)$, (x, y) are the Cartesian coordinates of the rear axis midpoint with respect to a reference frame, θ is the heading angle with respect to the frame x-axis, as illustrated in Figure 6.1. Let l be the inter-axle distance and ϕ be the steering angle between the average of front wheels and the main direction of the vehicle. The kinematic state equation is then given as

$$\begin{cases} \dot{x} = v\cos\theta \\ \dot{y} = v\sin\theta \\ \dot{\theta} = \dfrac{v}{l}\tan\phi = vk \end{cases} \tag{6.1}$$

where v is the longitudinal speed and $k := \tan\phi/l$ is the steering control input. The model is known as the Dubins car [58] if v is a positive constant. In this chapter, we consider our vehicle is allow to move backward and $v \in \{v_0,\ -v_0\}$ $(v_0 > 0)$, which is known as the Reeds-&-Shepp's car [92].

There is a maximum steering angle for most car-like vehicles, say $|\phi| \le \phi_{\max} < \pi/2$, then

$$|k| \le k_{\max} = \frac{\tan\phi_{\max}}{l} < \infty \tag{6.2}$$

where $R := 1/k_{\max}$ represents the minimum turning radius for the vehicle.

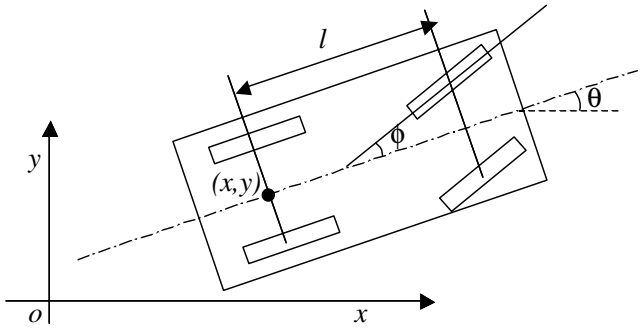


Figure 6.1: The car-like vehicle model.

The vehicle described in (6.1) is subject to the following nonholonomic constraint:

$$\dot{x}\sin\theta - \dot{y}\cos\theta = 0 \tag{6.3}$$

which mean that the vehicle is obliged to move tangentially to its main axis due to the assumption that side-slipping does not occur at low speeds.

6.2.2 The Vector Fields and the State Flow

For a car-like vehicle, there are two controls: the direction of the vehicle longitudinal speed v and the steering control k. In other words, we may move the vehicle forward or backward, or turn the steering wheel. These two controls form vector fields in the configuration space (C-space, or $\mathcal{C}$).

Equation (6.1) can be rewritten as follows

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \cos\theta \\ \sin\theta \\ 0 \end{pmatrix} v + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} vk \tag{6.4}$$

Let $f_1 := (\cos\theta \ \sin\theta \ 0)^T$ and $f_2 := (0 \ 0 \ 1)^T$, then the state equation can be written as

$$\dot{\xi} = (f_1 + f_2 k)v. \tag{6.5}$$

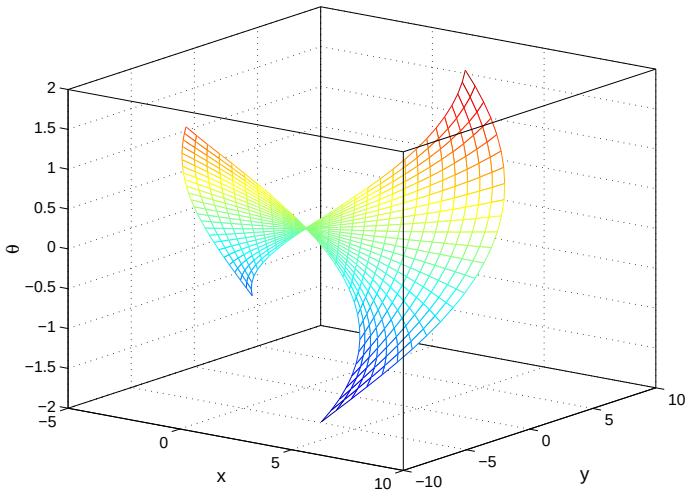


Figure 6.2: The state flow lines in the (x, y, θ)-space with various values of k.

So, for each fixed steering input k, $(g_k := f_1 + f_2 k)$ is a vector field. These vector fields form a vector space by linear combination and $\{f_1,\ f_2\}$ is a basis of the vector space.

Suppose the system state (ξ) is influenced by one of the vector fields, say g_k. Then, for any point $\xi_0 \in \mathcal{C}$, there is exactly one *state flow* $\phi_k(\tau)$ that starts from ξ_0 and follows g_k. That is, $\phi_k(0) = \xi_0$ and $\dot{\phi}_k(\tau) = g_k(\phi_k(\tau))$. Figure 6.2 shows the state flow lines in $\mathcal{C}$ that start from $(0,\ 0,\ 0)^T$, with various of v and k.

It is customary to define the state flow of the vector field g_k, as an *operator*, $\Phi_k(\tau)$, which acts on a state point ξ to produce the state flow line of the field, with $\Phi_k(0)\xi = \xi$. Therefore, the vehicle state trajectory is the orbit of the operators' actions on the vehicle state. The operators are written in the form of $\Phi(t; (v,\ k))$, for

example,

$$\Phi(t;(v,\ 0))\begin{pmatrix} x \\ y \\ \theta \end{pmatrix} = \begin{pmatrix} x + vt\cos\theta \\ y + vt\sin\theta \\ \theta \end{pmatrix} \tag{6.6}$$

and

$$\Phi(t;(v,\ k)\begin{pmatrix} x \\ y \\ \theta \end{pmatrix} = \begin{pmatrix} x + \frac{2}{k}\sin\frac{vkt}{2}\cos(\theta + \frac{vkt}{2}) \\ y + \frac{2}{k}\sin\frac{vkt}{2}\sin(\theta + \frac{vkt}{2}) \\ \theta + vkt \end{pmatrix} \tag{6.7}$$

where $v \in \{v_0,\ -v_0\}$ and $k \neq 0$.

6.2.3 The Unit Operations and Lie Algebra Operation

For an AGV, the maneuver plan consists of a series of *unit maneuver operations*, each of which specifies the *fixed* values $(v,\ k)$[13] for a unit time duration T.

Let $\mathcal{A}$ be the collection of all unit maneuver operations and let $a \in \mathcal{A}$ corresponding to $(v,\ k)$, then

$$a(\xi) = \Phi(T;(v,\ k))(\xi) \tag{6.8}$$

The inverse of a, a^{-1}, is defined by the one that operates at $(-v,\ k)$, so that $a \circ a^{-1} = a^{-1} \circ a = I_3$. Also, we define $\alpha a := \Phi(\alpha T;(v,\ k))$, for $\alpha \in \mathbf{R}^+$.

Note that the operators in $\mathcal{A}$ do not "commute" in general:

$$\exists a, b \in \mathcal{A}, s.t. : a \circ b \neq b \circ a$$

Applying the tools of *Lie algebra*, see [93, 94, 57] and the references therein, the previous vector fields can generate a new linearly independent vector field by concatenating infinitesimal motions:

$$c = b^{-1} \circ a^{-1} \circ b \circ a, \quad \text{for some} a, b \in \mathcal{A} \tag{6.9}$$

[13]Sometimes the front wheel steering control ϕ is specified instead, which is equivalent to k by $k = \tan\phi/l$.

where operation c means: first forward in one flow (a), then forward in the second flow (b), then backward in the first one (a^{-1}), and finally backward in the second one (b^{-1}). In general, $c \neq I_3$ because the operators are not commutative. For example, let $a = \Phi(\Delta T; (v_0,\ 0))$ and let $b = \Phi(\Delta T; (v_0,\ k))$ $(k \neq 0)$, then the action of c on $\xi = (x, y, \theta)^T \in \mathcal{C}$ can be calculated from (6.6) and (6.7):

$$
\begin{aligned}
c(\xi) &= b^{-1} \circ a^{-1} \circ b \circ a(\xi) \\
&= b^{-1} \circ a^{-1} \circ b \left(\begin{bmatrix} x + v_0 \Delta T \cos\theta \\ y + v_0 \Delta T \sin\theta \\ \theta \end{bmatrix} \right) \\
&= b^{-1} \circ a^{-1} \left(\begin{bmatrix} x + v_0 \Delta T \cos\theta + \frac{2}{k} \sin\frac{v_0 k \Delta T}{2} \cos(\theta + \frac{v_0 k \Delta T}{2}) \\ y + v_0 \Delta T \sin\theta + \frac{2}{k} \sin\frac{v_0 k \Delta T}{2} \sin(\theta + \frac{v_0 k \Delta T}{2}) \\ \theta + v_0 k \Delta T \end{bmatrix} \right) \\
&= b^{-1} \left(\begin{bmatrix} x + v_0 \Delta T \cos\theta + \frac{2}{k} \sin\frac{v_0 k \Delta T}{2} \cos(\theta + \frac{v_0 k \Delta T}{2}) - v_0 \Delta T \cos(\theta + v_0 k \Delta T) \\ y + v_0 \Delta T \sin\theta + \frac{2}{k} \sin\frac{v_0 k \Delta T}{2} \sin(\theta + \frac{v_0 k \Delta T}{2}) - v_0 \Delta T \sin(\theta + v_0 k \Delta T) \\ \theta + v_0 k \Delta T \end{bmatrix} \right) \\
&= \begin{bmatrix} x + v_0 \Delta T \cos\theta - v_0 \Delta T \cos(\theta + v_0 k \Delta T) \\ y + v_0 \Delta T \sin\theta - v_0 \Delta T \sin(\theta + v_0 k \Delta T) \\ \theta \end{bmatrix} \\
&= \begin{bmatrix} x + 2 v_0 \Delta T \sin\frac{v_0 k \Delta T}{2} \sin(\theta + \frac{v_0 k \Delta T}{2}) \\ y - 2 v_0 \Delta T \sin\frac{v_0 k \Delta T}{2} \cos(\theta + \frac{v_0 k \Delta T}{2}) \\ \theta \end{bmatrix} \\
&\to \begin{bmatrix} x + d\sin(\theta) \\ y - d\cos(\theta) \\ \theta \end{bmatrix}, \quad \text{as} \quad \Delta T \to 0, \quad \text{and} \quad d := v_0^2 k \Delta T^2.
\end{aligned}
\tag{6.10}
$$

Therefore, the combined operation $c = b^{-1} \circ a^{-1} \circ b \circ a$ virtually represents a new vector field $f_3 := \begin{bmatrix} \sin\theta \\ -\cos\theta \\ 0 \end{bmatrix}$, which is linearly independent to the basis of the original vector space: $f_1 = \begin{bmatrix} \cos\theta \\ \sin\theta \\ 0 \end{bmatrix}$ and $f_2 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$. In fact, the new vector field allows the vehicle to "slide" laterally without changing the vehicle's heading, as shown in Figure 6.3.

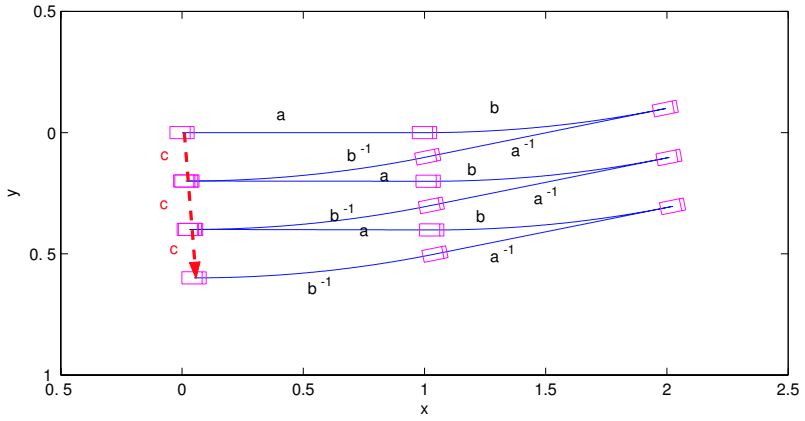


Figure 6.3: An example that illustrates the Lie algebra action.

6.3 Maneuver Automation Problem Setup

Problem 6.1 The maneuver automation problem in free space: *Given any two points in the C-space, say* $\xi_0, \xi_g \in \mathcal{C}$*, find a sequence of unit operators* $a_1, a_2, \cdots, a_N \in \mathcal{A}$*, each* a_i *is corresponding to a fixed values of* $(v_i,\ k_i)$*, and the corresponding time scaling factors* $\alpha_1, \alpha_2, \cdots, \alpha_N \in \mathbf{R}^+$*, such that*

$$d(a(\xi_0), \xi_g) < \varepsilon$$

where d *is a metric defined in* $\mathcal{C}$*,* $\varepsilon > 0$ *is a given value, and the operator* a *is defined by*

$$a = (\alpha_N a_N) \circ \cdots \circ (\alpha_2 a_2) \circ (\alpha_1 a_1)$$

.

The problem is not only solvable but also has infinite number of solutions, among which we want find a solution that has the minimum the state flow distance from ξ_0 to the ε-neighbourhood of ξ_g. If we fix that $|v| = v_0 > 0$, the optimization problem is

equivalent to the minimum-time problem. That is, we want to find the one that

$$\text{minimize} \quad J = \sum_{i=1}^{N} \alpha_i \tag{6.11}$$

For a vehicle that can only move forward and has constraint on its turning radius, the problem is still solvable and has infinite number of solutions too. The initial work was done by Dubins from 1957 [58]. He proved the existence of shortest paths and showed that each of the unit operations in an optimal solution has to be either $\Phi(t; (v_0,\ 0))$ (type "S", for straight line segments) or $\Phi(t; (v_0,\ \pm k_{\max}))$ (type "C", for arcs of circle with radius R). Furthermore, Dubins showed that optimal paths are necessarily made with at most three ($N \leq 3$) pieces of such segments and they belong to the following sufficient family:

$$\{CCC,\ CSC\}.$$

Later, Sussmann and Tang [95] and Boissonnat, Cerezo and Leblond [96] gave new proof of Dubins result using optimal control theory.

In 1990, Reeds and Shepp [92] considered the same problem that allows backwards motions of the vehicle. From Dubins result, they proved that the shortest paths are restricted to

$$\{C|C|C,\ CC|C,\ C|CC,\ CC|CC,\ C|CC|C,\ C|CSC,\ CSC|C,\ C|CSC|C,\ CSC\}$$

where "|" indicates the sign change in vehicle speed.

We shall consider the maneuver automation problem with obstacles around the vehicle. The C-space is partitioned into $\mathcal{C}_{free}$ and $\mathcal{C}_{obs}$ subsets. We say the vehicle collides an obstacle if its state $\xi \in \mathcal{C}_{obs}$.

Problem 6.2 The maneuver automation problem among obstacles: *Given any two points in the C-space, say $\xi_0, \xi_g \in \mathcal{C}_{free}$, find a sequence of unit operators $a_1, a_2, \cdots, a_N \in \mathcal{A}$, each a_i is corresponding to a fixed values of $(v_i,\ k_i)$, and the corresponding time scaling factors $\alpha_1, \alpha_2, \cdots, \alpha_N \in \mathbf{R}^+$, that 1) solve Problem 6.1 and 2) the flow path from ξ_0 to $a(\xi_0)$ is contained in $\mathcal{C}_{free}$.*

6.4 Direct Approach by Using Lie Algebra Tools

Problem 6.2 concerns finding a sequence of maneuver operations that drive the vehicle state from an initial configuration to a goal configuration while avoiding obstacles. Since a Reeds-&-Shepp's car is allow to flow in the vector space that has a full-ranked basis $\{f_1,\ f_2,\ f_3\}$, the vehicle is "virtually" allowed to move in any direction in the C-space. Hence, the problem can be solved in two steps:

1. Compute collision-free paths without considering the kinematic constraints. We may use certain geometric path planners to generate a collision-free path, if one exists. The path is then admissible for a holonomic system. If no such path exists, then there is no solution.

2. Compute unit maneuver plans at each non-smooth points where the state flow is not in $span\{f_1,\ f_2\}$ by using Lie algebra tools as in (6.10) that allow the vehicle to move like a holonomic system.

The above approach uses Lie algebra approximation method to "convert" the nonholonomic system into a holonomic system which does not have any kinematic constraints and can move in any directions from any configuration points. Therefore, this holonomic approximating approach works in principle. However, the implementation of "virtual" motions that concatenates a series of infinitesimal motions is very

time consuming and far from optimal due to the different orders of magnitude involved. So, this method is adoptable only if the total distance for lateral "shifting" is not too long.

In the following, we will use the dynamic programming method and the nonlinear control approach to generate maneuver plans that avoids obstacles.

6.5 Dynamic Programming Method

6.5.1 Formulating the Planning Process

Using Dynamic Programming (DP) [97], it is theoretically possible to find an optimal policy. In our problem, the optimal policy means the maneuver flow path that is shortest (and safest) to connect from ξ_0 to ξ_g. Suppose we only concern a plan with N step, define $J_m(\xi_m)$ as the "cost-to-go" at the mth step. Then theoretically, the optimal plan can be obtained by taking the arguments of the minimization of the DP recursion

$$J_m(\xi_m) = \min_{a \in \mathcal{A}} \{g(\xi_m, a) + J_{m+1}(a(\xi_m))\} \tag{6.12}$$

In this equation, $\mathcal{A}$ is the collection of all admissible operators. $g(\xi_m, a)$ is the cost in one plan step, for example, the value can be the curve distance from ξ to $a(\xi)$, or the value can be related to the distance from the curve to nearest obstacles, etc. Note that $\xi_{m+1} = a(\xi_m)$, which produces the next state from the current state and control.

We also define the following properties so that the (recursive) search can be terminated.

- If $d(\xi_m,\ \xi_g) < \varepsilon$, then $J_m(\xi_m) = 0$.

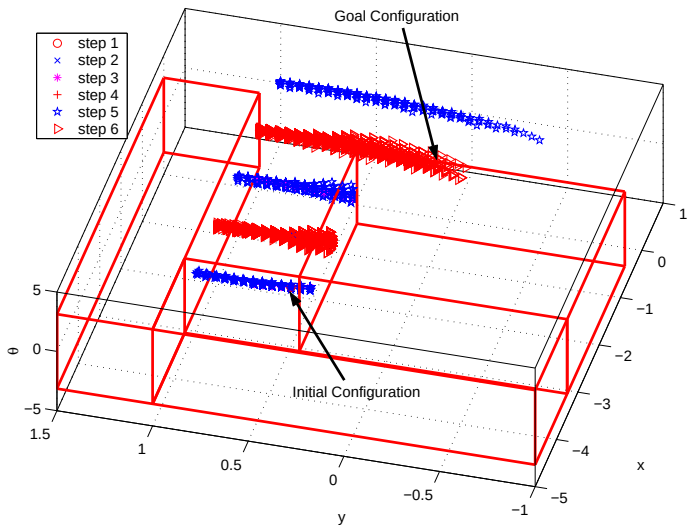


Figure 6.4: Illustration of DA method. The initial configuration is at (-2, 0.5, 0) and goal configuration is at (0, 0, 0). $\mathcal{C}_{obs}$ is shown as boxes.

- If $\xi_m \in \mathcal{C}_{obs}$, then $J_m(\xi_m) = \infty$, so that the states inside obstacles are not eligible for search.

- If a state flow path from ξ_m to $a(\xi_m)$ intersect with $\mathcal{C}_{obs}$, then $g(\xi_m, a) = \infty$.

Figure 6.4 shows an example of DA method, where $\xi_0 = (-2, 0.5, 0)$ and $\xi_g = (0, 0, 0)$. We use a very rough sampling ($T = 1,\ v \in \{-1,\ 1\}, k \in \{-0.2,\ 0,\ 0.2\}$) so that we can produce a presentable result. Still, after six step, the result positions in six step can be as large as 6^6. Although we reduced a lot by invalidating states that fall into $\mathcal{C}_{obs}$, the number of remained states is still very large.

6.5.2 Rolling Horizon Approximation

However, attempting to solve equation (6.12) for the optimal cost-to-go produces a very complex problem, because the "curse of dimensionality" causes an explosion of possible states to examine over the entire planning horizon. When the total number

of plan steps N is large, which is usually true, the DA method is not admissible at all.

A *rolling horizon approximation* (RHA) may produce a feasible result at a cost of optimality. In RHA method, a short planning span is defined, say a small value of H is selected. After H steps, the configuration ξ_H may not be in the ε-ball of the goal configuration ξ_g, so the value of $J_H(\xi_H)$ is not zero and we don't have means to calculate it quickly. However, we can substitute $J_H(\xi_H)$ by $\tilde{J}(\xi_H)$ by reasonable approximations. Then, the RHA method is to solve the following:

$$\begin{aligned} J(\xi_0) = \min_{a_1 \in \mathcal{A}} \{ g(\xi_0, a_1) + \min_{a_2 \in \mathcal{A}} \{ g(\xi_1, a_2) + \cdots \\ + \min_{a_{H-1} \in \mathcal{A}} \{ g(\xi_{H-1}, a_{H-1}) + \tilde{J}(\xi_H)) \} \} \} \end{aligned} \tag{6.13}$$

In RHA method, the problem space can be much smaller than the original one because H can be chosen such that $H \ll N$. However, this method may not produce the optimal solution to the original problem. Another drawback in this situation is the difficulty in finding the approximate function $\tilde{J}\xi_H$. If not properly selected, the RHA may lead to local obstacle trap and get stuck.

6.5.3 Best First Search

By defining the configurations that a vehicle would travel to at each step as the nodes of a graph, we can generate a graph in tree formation. The root is ξ_0 and very possible locations that the vehicle could reach are the nodes (including the leaves). The numbers of layers of branches of the tree is equal to the search depth (i.e. N).

We can label each node with an approximate value to the goal and adopt the best-first-search (BFS) algorithms, such as A* algorithm [45] or Dijkstra's algorithm [98]. In these methods, the best available node would always be searched first and the algorithm would traverse this node before the others. If the approximate values

being properly selected, then the first path that reaches the termination node is the best path.

6.6 Nonlinear Controller

In this section, we shall propose a nonlinear feedback controller to generate the control commands v and k. Without losing generality, we concern driving the system $\mathbf{\Sigma}$ (first defined in (6.1)):

$$\begin{cases} \dot{x} = v\cos\theta \\ \dot{y} = v\sin\theta \\ \dot{\theta} = vk \end{cases} \tag{6.14}$$

from $\xi_0 \in \mathcal{C}$ to the origin with $v \in \{v_0, -v_0\}$ and $|k| \leq k_{\max}$.

The maneuver process consists of two steps:

1. Drive the state to a manifold: $\mathcal{M} := \{\xi = (x, y, \theta)^T || y| \leq \epsilon_1 \text{ and } |\theta| \leq \epsilon_2\}$, where $\epsilon_1,\ \epsilon_2 > 0$ are small values.

2. Drive x to 0 by selecting proper speed direction and let $k = 0$. If the state escapes the manifold, go to step 1.

In step 1, we first let $v = v_0$ and let

$$k = -\lambda_1 \frac{\theta}{v} - \lambda_2 \frac{\sin\theta}{\theta} y. \tag{6.15}$$

where $\lambda_1,\ \lambda_2 > 0$.

Proposition 6.1 *System $\mathbf{\Sigma}$ (6.14) globally converges to subspace $\{\xi | y = 0,\ \text{and } \theta = 0\}$ (i.e. the x-axis in $\mathcal{C}$) under the nonlinear steering controller k that is defined in (6.15).*

Proof: Let

$$V(y,k) = \frac{1}{2}(\lambda_2 y^2 + \theta^2)$$

be a candidate Lyapunov function, which is positive definite and radically unbounded.

Then,

$$\begin{aligned}\dot{V} &= \lambda_2 y\dot{y} + \theta\dot{\theta} \\ &= \lambda_2 yv\sin\theta + \theta v(-\lambda_1\frac{\theta}{v} - \lambda_2\frac{\sin\theta}{\theta}y) \\ &= -\lambda_1\theta^2 \leq 0\end{aligned}$$

From LaSalle's Theorem, $(y,\ k)$ converges to the limit set $\omega := \{(y,k)|\dot{V}(y,k) = 0\}$, which requires that both $\theta = 0$ and $\dot{\theta} = 0$. The latter implies that $k = 0$. From (6.15) and $\theta = 0$, we have $y = 0$.

Therefore, system $\mathbf{\Sigma}$ globally converges to subspace $\{\xi|y = 0,\ \text{and}\ \theta = 0\}$. ■

As far as obstacles are concerned, we use following strategy: whenever the following direction may lead to collision with obstacles, switch the direction of the vehicle speed, i.e. let $v(t) = -v(t^-)$ and then recalculate the k. Note that in the previous proof, neither the Lyapunov function V nor the proof itself relies on the value v as long as $v \neq 0$. Therefore, the system still converges to manifold $\mathcal{M}$, if not stuck by an obstacle[14]. Also, due to the maximum curvature constraint, we saturate our steering control k by $\hat{k} = k_{\max}\text{SAT}(k/k_{\max})$, where

$$\text{SAT}(x) = \begin{cases} -1, & \text{if } x < -1, \\ x, & \text{if } -1 \leq x \leq 1, \\ 1, & \text{if } x > 1 \end{cases}$$

In step 2, when $\xi \in \mathcal{M}$, i.e. $|y| \leq \epsilon_1$ and $|\theta| \leq \epsilon_2$, we apply $v = -v_0\text{sgn}(x)$ and $k = 0$, where

$$\text{sgn}(x) = \begin{cases} -1, & \text{if } x < 0, \\ 0, & \text{if } x = 0, \\ 1, & \text{if } x > 0 \end{cases}$$

[14]It is possible that the system be stuck before entering the manifold.

So, in the manifold, the state equation can be approximated by $\dot{x} \simeq v = -v_0 \text{sgn}(x)$. Then x converges to 0 as long as the system stays in the manifold.

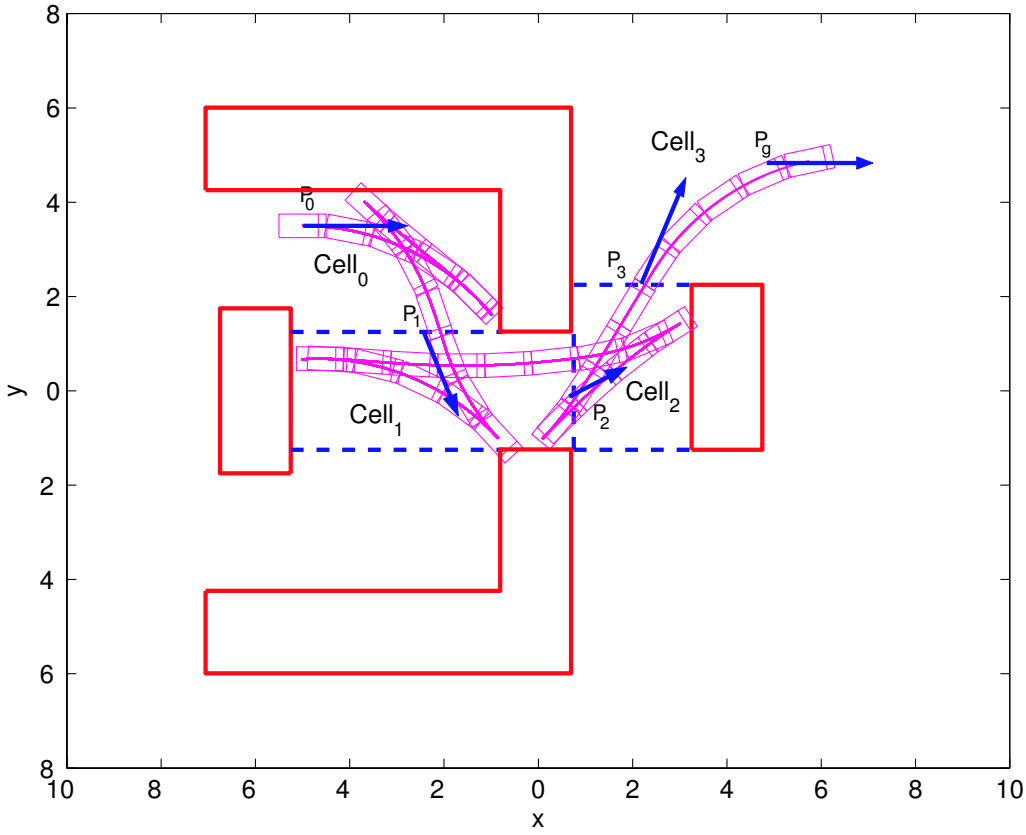
Now we have state feedback nonlinear controller that can maneuver the system state from one configuration to another. We adopt the idea of cell decomposition method [6, 35] to partition the $\mathcal{C}_{free}$ into finite number of convex cells, and construct the connectivity graph associated with the cells, and then search the graph for a sequence of adjacent cells from the cell that contains the initial point to the one that contains the goal point. Finally, we obtain a series of "waypoints", each of which stays in the middle of the cell border. Since the nonlinear controller does not have problem in maneuver the system state between any two configurations that are contained in the same convex region. Applying the nonlinear controller to follow the waypoints one by one solves the maneuver problem.

Figure 6.5 shows the simulation result of automatic maneuver with the nonlinear controller with convex cell decomposition.

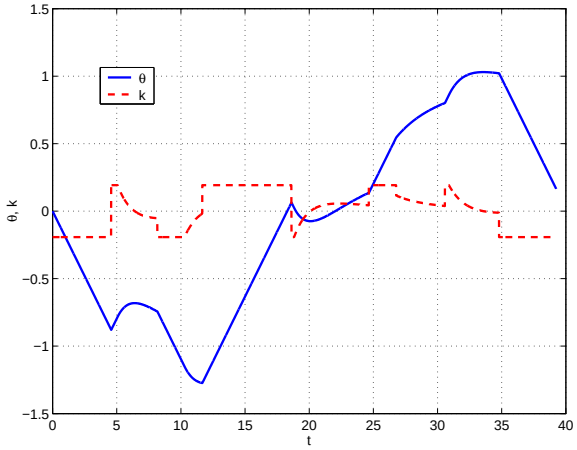
6.7 Conclusion

In this chapter, we discuss the maneuver automation problem of driving a nonholonomic vehicle with constraints on turning radius from one configuration to another.

After brief introduction to the kinematic system model of a car-like vehicle, we construct the vector fields and the state flows of the system in the C-space. We then define the unit maneuver operations. By introducing the Lie algebra operator among these unit maneuver operations, we show that by repeating interleaving and back-and-forth motions of infinitesimal time duration, one can generate a new vector field that is linearly independent to the existing vector fields. Furthermore, the new vector



a. The trajectory of vehicle path.



b. The vehicle heading θ and curvature k.

Figure 6.5: A simulation result of the nonlinear controller with convex cell decomposition. P_0 is the original position, P_g is the goal position, P_1, P_2, P_3 are the intermediate waypoints. The arrow on P_0 indicates the original vehicle heading, and the other arrows indicate the desired heading.

"virtually" allows vehicle to "slide" laterally without changing the vehicle's direction, which leads to the direct approach of maneuver planning: by holonomic planning and "converting" the nonholonomic system into a holonomic system by means of Lie algebra approximation. Because the implementation of "virtual" motions may be time consuming and not optimal in many situations, this method is suitable only when the distance for lateral sliding is not too long.

We then introduce the dynamic programming method, which is theoretically possible to find an optimal maneuver flow path. Due to the "curse of dimensionality", the dynamic programming method is not applicable directly. We then present the rolling-horizon method and best-first-search method to reduce the computation requirement when we can estimate the cost from any configuration to the goal.

A nonlinear control method is proposed for automatic maneuver. It is proved, by LaSalle's Theorem, that the system globally converges to the x-axis under the nonlinear feedback control. In other words, the origin in the (y, θ)-plane is globally asymptotically stable for the projected system onto the (y, θ)-plane. In the simulation, it is shown that the nonlinear control method, combined with the geographical cell decomposition method, is able to maneuver the vehicle from one configuration to another amidst obstacles.

CHAPTER 7

CONCLUSIONS

7.1 Summary and Contribution

In this dissertation, we study the structure and algorithms in developing the control system of an autonomous ground vehicle. It consists of two major parts.

In the first part, we propose a hierarchical generic control system architecture that is suitable for developing a class of AGVs. It satisfies the general requirements for an AGV control system. The whole functionalities that are required for the control system have been carefully decomposed into smaller tasks which are then delegated into the several components in the architecture. This hierarchical structure alleviates the conflicts between the high complexity and limited computing resource. The functionalities of each component in the structure and the interconnections among these components are described in Chapter 2. As an example, ION, the autonomous vehicle demonstrated in DARPA Grand Challenge 2005, is introduced in Chapter 3. Its control system structure and the hierarchical task decomposition are studied and designed. The discrete-event state machine in ION's navigation module, as well as the physical layout of the vehicle hardware, is also discussed in this chapter.

In the second part, we focus on the algorithms that are applied in the navigation module to generate safe, efficient and feasible local path from the current position to the specified goals in real time. In Chapter 4, we propose a real-time path-planning algorithm utilizing fuzzy logic. The fuzzy rules are designed and applied to a basic steering strategy. Not only the simulation but also its practical implementations justify its performance. In Chapter 5, the problem of smooth path-planning is defined and studied. The smooth path-planning problem is formulated as a second-order geometric continuous (smooth) interpolation problem and a piecewise smooth interpolation problem. We then propose to use the quartic polynomials to solve the latter problem with only local waypoint positions. The minimum offset smooth interpolation problem is the formulated and studied. With the given way points and maximum turning curvature, the minimum offset can be calculated by using quartic splines. Chapter 6 studies the problem of automatic maneuver planning for car-like vehicles which are nonholonomic systems with constraints on their turning radius. We systematically define the vector space and state flow of the nonholonomic system. Lie algebra is applied to generate a "virtual" vector field, which leads to the direct approach of maneuver planning. Dynamic programming method is also introduced to the planning problem. Finally, we also propose a nonlinear control method for automatic maneuver. It is proved, by LaSalle's Theorem, that the system globally converges to the x-axis under the nonlinear feedback control. By simulations, we have shown that with the nonlinear control and the geographical cell decomposition method, the maneuver plans can be generated to connect one configuration to another.

7.2 Future Research Directions

There are lots of possible future research directions in the control systems of autonomous ground vehicles, such as reinforcement learning and adaption to the environment, map-based route planning and automatic replanning, human-vehicle and inter-vehicle cooperation, and traffic rule enforcement, etc.

There are several possible improvements that can be made to the current control system of ION. We could implement "advanced" situation recognition that not only detects current situation but also anticipates the situations that the vehicle might get into in near future, so that the navigator can take precautions against very difficult situations. The obstacle avoidance algorithm could be improved to deal with moving obstacles by anticipating their positions in the future. We could also adopt other path-planing algorithms to coexist in the navigation module. At the expense of computing resource, these algorithms generate possibly different paths, among which we could select the one that fits the situation most. Last, but not least, we could implement the roll-back functionality which, at the expense of storage space, records the past trajectory that the vehicle has covered and when the vehicle gets stuck, it drives the vehicle back along the recorded path to a point from where another path can be initiated.

BIBLIOGRAPHY

[1] M. Daily, J. Harris, D. Keirsey, K. Olin, D. Payton, K. Reiser, J. Rosenblatt, D. Tseng, and V. Wong, "Autonomous cross-country navigation with the ALV," in *IEEE Conference on Robotics and Automation*, (Philadelphia, PA), pp. 718–726, Apr. 1988.

[2] K. E. Olin and D. Y. Tseng, "Autonomous cross-country navigation: An integrated perteption and planning system," *IEEE Expert*, vol. 6(4), pp. 16–30, Aug. 1991.

[3] C. Shoemaker and J. Bornstein, "The demo III UGV program: a testbed for autonomous navigation research," in *ISIC/CIRA/ISAS Joint Conference*, (Gaithersburg, MD), pp. 644–651, 1998.

[4] Grand Challenge Website. [online] http://www.grandchallenge.org.

[5] US GAO, "Defense acquisition: Army transformation faces weapon systems challenges," 2001. [online] http://www.gao.gov/new.items/d01311.pdf.

[6] N. Nilsson, "A mobile automation: An application of artificial intelligence techniques," in *1st Int. Joint Conf. on Artificial Intelligence*, (Washington DC), pp. 509–520, 1969.

[7] Q. Chen and Ü. Özgüner, "Intelligent off-road navigation algorithms and strategies of team desert buckeyes in the darpa grand challenge '05," *Journal of Field Robotics*, vol. 23, pp. 729–743, Sept. 2006.

[8] Q. Chen, Ümit. Özgüner, and K. Redmill, "The Ohio State University team at the grand challenge 2004: Developing a completely autonomous vehicle," *IEEE Intelligent Systems*, vol. 19, pp. 8–11, Sept.-Oct. 2004.

[9] Ü. Özgüner, "Coordination of hierarchical systems," in *Proc. IEEE International Symposium on Intelligent Control*, (Philadelphia, PA), pp. 2–7, Sept. 1990.

[10] L. Acar and Ü. Özgüner, "Design of knowledge-rich hierarchical controllers for large functional systems," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 20, pp. 791–803, July 1990.

[11] L. Acar and Ü. Özgüner, "Design of structure-based hierarchies for distributed intelligent control," in *An Introduction to Intelligent and Autonomous Control Systems* (P. J. Antsaklis and K. M. Passino, eds.), pp. 79–108, Boston, MA: Kluwer Academic Publishers, 1993.

[12] K. Redmill, Ü. Özgüner, J. Musgrave, and W. Merrill, "Intelligent hierarchical thrust vector control for a space shuttle," *IEEE Control Systems Magazine*, vol. 14, pp. 13–23, June 1994.

[13] D. Clark, "HIC: An operating system for hierarchies of servo loops," in *Robotics and Automation, IEEE International Conference on*, vol. 2, (Scottsdale, Az), pp. 1004–1009, May 1989.

[14] R. Murray, D. Deno, K. Pister, and S. Sastry, "Control primitives for robot systems," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 22, pp. 183–193, Jan. 1992.

[15] J. K. Rosenblatt, "DAMN: A distributed architecture for mobile navigation," in *Proc. of the AAAI Spring Symp. on Lessons Learned from Implememted Software Architectures for Physical Agents*, (Stanford, CA), 1995.

[16] DGC05 Teams. [online] http://www.darpa.mil/grandchallenge/TechPapers.

[17] J. S. Albus, "4-D/RCS reference model architecture for unmanned ground vehicles," in *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, (San Francisco, CA), Apr. 2000.

[18] M. Moore, V. Gazi, K. Passino, F. Proctor, and W. Shackleford, "Complex control system design and implementation using the NIST-RCS software library," *IEEE Control Systems Magazine*, vol. 19, pp. 12, 14–28, Dec. 1999.

[19] JAUS. [online] http://www.jauswg.org/baseline/refarch.html.

[20] M. S. Branicky, *Studies in Hybrid Systems: Modeling, Analysis, and Control*. PhD thesis, MIT, Cambridge, Mass, 1995.

[21] A. Morse, ed., *Control Using Logic-Based Switching*. Berlin, Germany: Springer-Verlag, 1996.

[22] in *Special issue on hybrid systems: theory and applications, Proceedings of the IEEE*, July 2000.

[23] E. Frazzoli, *Robust Hybrid Control for Autonomous Vehicle Motion Planning*. PhD thesis, MIT, Cambridge, Mass, 2001.

[24] T. Lozano-Pérez, "Spatial planning: a configuration space approach," *IEEE Trans. Computers*, vol. 32(2), pp. 108–120, Feb. 1983.

[25] J. T. Schwartz and M. Sharir, "On the piano movers' problem II: General techniques for computing topological properties of real algebraic manifolds," *Advances in Applied Math*, vol. 4, pp. 298–351, 1983.

[26] J.-C. Latombe, *Robot motion planning*. Boston : Kluwer Academic Publishers, 1991.

[27] J.-C. Latombe, "Motion planning: A journey of robots, molecules, digital actors, and other artifacts," *International Journal of Robotics Research*, vol. 18, pp. 1119–1128, 1999.

[28] N. Rao, S. Iyengar, and G. deSaussure, "The visit problem: visibility graph-based solution," in *Proceedings of the 1988 IEEE International Conference on Robotics and Automation*, vol. 3, (Philadelphia, PA), pp. 1650–1655, Apr. 1988.

[29] M. Neus and S. Maouche, "Motion planning using the modified visibility graph," in *Systems, Man, and Cybernetics, IEEE International Conference on*, vol. 4, (Tokyo, Japan), pp. 651–655, Oct. 1999.

[30] O. Takahashi and R. Schilling, "Motion planning in a plane using generalized Voronoi diagrams," *Robotics and Automation, IEEE Transactions on*, vol. 5, pp. 143–150, Apr. 1989.

[31] H. Choset and J. Burdick, "Sensor based planning. I. the generalized Voronoi graph," in *Proceedings of the 1995 IEEE International Conference on Robotics and Automation*, vol. 2, (Nagoya, Japan), pp. 1649–1655, May 1995.

[32] H. Choset and J. Burdick, "Sensor based planning. II. incremental construction of the generalized Voronoi graph," in *Proceedings of the 1995 IEEE International Conference on Robotics and Automation*, vol. 2, (Nagoya, Japan), pp. 1643–1648, May 1995.

[33] L. E. Kavralu, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, Aug. 1996.

[34] L. Kavraki, M. Kolountzakis, and J.-C. Latombe, "Analysis of probabilistic roadmaps for path planning," in *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, vol. 14, pp. 166–171, Feb. 1998.

[35] G. Giralt, R. Sobek, and R. Chatila, "A multi-level planning and navigation systems for a mobile robot: a first approach to Hilare," in *6th Int. Joint Conf. on Artificial Intelligence*, (Tokyo, Japan), pp. 335–337, 1979.

[36] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Proceedings of the 1985 IEEE International Conference on Robotics and Automation*, vol. 2, pp. 500–505, Mar. 1985.

[37] C. Warren, "Global path planning using artificial potential fields," in *Proceedings of the 1989 IEEE International Conference on Robotics and Automation*, vol. 1, pp. 316–321, May 1989.

[38] J. Guldner and V. Utkin, "Sliding mode control for gradient tracking and robot navigation using artificial potential fields," *Robotics and Automation, IEEE Transactions on*, vol. 11, pp. 247–254, Apr. 1995.

[39] J. Barraquand and J.-C. Latombe, "Robot motion planning: A distributed representation approach," *International Journal of Robotics Research*, pp. 628–649.

[40] K. M. Passino, *Biomimicry for Optimization, Control, and Automation*. London, UK: Springer-Verlag, 2005.

[41] C.-H. Lin and L.-L. Wang, "Intelligent collision avoidance by fuzzy logic control," *Robotics and Autonomous Systems*, vol. 20, pp. 61–83, 1997.

[42] N. Hodge and M. Trabia, "Steering fuzzy logic controller for an autonomous vehicle," in *Proc. IEEE International Conference on Robotics and Automation*, (Detroit, MI), pp. 2482–2488, May 1999.

[43] S. Pawlowski, P. Dutkiewicz, K. Kozlowski, and W. Wroblewski, "Fuzzy logic implementation in mobile robot control," in *Robot Motion and Control, 2001 Proceedings of the Second International Workshop on*, pp. 65–70, Oct. 2001.

[44] A. Stentz, "Optimal and efficient path planning for partially-known environments," in *IEEE International Conference on Robotics and Automation (ICRA'94)*, May 1994.

[45] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, pp. 100–107, 1968.

[46] V. J. Lumelsky and A. A. Stepanov, "Dynamic path planning for a mobile automaton with limited information on the enviroment," *IEEE transactions on Automatic control*, pp. 1058–1063, 1986.

[47] V. J. Lumelsky, "Algorithmic issues of sensor-based robot motion planning," in *Proceedings of the 26th Conference on Decision and Control*, (Los Angeles, CA), pp. 1796–1801, 1987.

[48] J. Reif, "Complexity of the mover's problem and generalization," in *Proc. 20th IEEE Symp.*, (Orlando, FL), July 2002.

[49] J. F. Canny, *The Complexity of Robot Motion Planning.* PhD thesis, MIT, Cambridge, Mass, 1987.

[50] B. Brummit, R. C. Coulter, A. Kelly, and A. Stentz, "A system for autonomous cross country navigation," in *IFAC Symposium on Intelligent Components and Instruments for Control Applications*, (Malaga, Spain).

[51] J. Laumond, "Feasible trajectories for mobile robots with kinematic and environment constraints," in *Intelligent Autonomous Systems* (L. Hertzberger and F. Groen, eds.), pp. 346–354, New York: North-Holland, 1987.

[52] J. Laumond, "Finding collision-free smooth trajectories for a non-holonomic mobile robot," in *10th Int. Joint Conf. on Artificial Intelligence*, (Milan), pp. 1120–1123, Aug. 1987.

[53] J. Laumond, "Singularities and topological aspects in nonholonomic motion planning," in *Nonholonomic Motion Planning* (Z. Li and J. F. Canny, eds.), Kluwer Academic Publishers, 1992.

[54] Z. Li and J. F. Canny, eds., *Nonholonomic Motion Planning.* Kluwer Acadamic Publisher, 1992.

[55] J. Laumond, P. Jacobs, M. Taix, and R. M. Murray, "A motion planner for nonholonomic mobile robots," *IEEE Trans. on Robotics and Automation*, vol. 10, pp. 577–593, Oct. 1994.

[56] A. M. Bloch, *Nonholonomic Mechanics and Control.* Berlin: Springer-Verlag, 2003.

[57] S. M. LaValle, *Planning Algorithms.* Cambridge University Press, 2006. Also available [online] http://msl.cs.uiuc.edu/planning/.

[58] L. E. Dubins, "On curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents," *American Journal of Mathematics*, 1957.

[59] J. Ibarra-Zannatha, J. Sossa-Azuela, and H. Gonzalez-Hernandez, "A new roadmap approach to automatic path planning for mobile robot navigation," in *Systems, Man, and Cybernetics, 1994 IEEE International Conference on*, pp. 2803–2808, Oct. 1994.

[60] A. Simon and J. Becker, "Vehicle guidance for an autonomous vehicle," in *Intelligent Transportation Systems, Proceedings of 1999 IEEE/IEEJ/JSAI International Conference on*, pp. 429–434, Oct. 1999.

[61] K. Nagatani, Y. Iwai, and Y. Tanaka, "Sensor based navigation for car-like mobile robots using generalized Voronoi graph," in *Intelligent Robots and Systems, Proceedings of the 2001 IEEE/RSJ International Conference on*, vol. 2, pp. 1017–1022, Oct. 2001.

[62] A. Piazzi and C. Guarino Lo Bianco, "Quintic G^2-splines for trajectory planning of autonomous vehicles," in *Intelligent Vehicles Symposium, 2000. Proceedings of the IEEE*, pp. 198–203, Oct. 2000.

[63] Y. Kanayama and B. Hartman, "Smooth local path planning for autonomous vehicles," in *Proceedings of the 1989 IEEE International Conference on Robotics and Automation*, vol. 3, pp. 1265–1270, May 1989.

[64] Y. Kanayama and N. Miyake, "Trajectory generation for mobile robots," *Robotic Research*, pp. 333–340, 1986.

[65] C. Guarino Lo Bianco and A. Piazzi, "Optimal trajectory planning with quintic G^2-splines," in *Intelligent Vehicles Symposium, 2000. Proceedings of the IEEE*, pp. 620–625, Oct. 2000.

[66] A. Piazzi, C. Guarino Lo Bianco, M. Bertozzi, A. Fascioli, and A. Broggi, "Quintic G^2-splines for the iterative steering of vision-based autonomous vehicles," *Intelligent Transportation Systems, IEEE Transactions on*, pp. 27–36, Mar. 2002.

[67] K. Redmill, J. Martin, and Ümit. Özgüner, "Sensing and sensor fusion for the 2005 desert buckeyes darpa grand challenge offroad autonomous vehicle," in *IEEE Intelligent Vehicle Symposium (IV '06)*, (Tokyo, Japan), June 2006.

[68] ION. http://www.darpa.mil/grandchallenge/TechPapers/DesertBuckeyes.pdf.

[69] B. Hummel, S. Kammel, T. Dang, C. Duchow, and C. Stiller, "Vision-based path planning in unstructured environments," in *IEEE Intelligent Vehicle Symposium (IV '06)*, (Tokyo, Japan), June 2006.

[70] K. A. Redmill, *Automated Vehicles: The Nature and Implementation of Autonomous Multi-Agent Systems*. PhD thesis, The Ohio State University, Columbus, OH, 1998.

[71] G. Walsh, D. Tilbury, S. Sastry, R. Murray, and J. Laumond, "Stabilization of trajectories for systems with nonholonomic constraints," *IEEE Transactions on Automatic Control*, vol. 39(1), pp. 216–222, Jan. 1994.

[72] K. Redmill and Ü. Özgüner, "The Ohio State University automated highway system demonstration vehicle," in *SAE Paper 980855, 1998SAE International Congress and Exposition*, (Detroit, MI), pp. 117–130, Feb. 1998.

[73] C. Hatipoglu, Ü. Özgüner, and K. Redmill, "Automated lane change controller design," *IEEE Transactions on Intelligent Transportation Systems*, vol. 4, pp. 13–22, Mar. 2003.

[74] U. Kiencke and L. Nielsen, *Automotive Control Systems for Engine, Driveline, and Vehicle*. Springer, 2000.

[75] K. Redmill, T. Kitajima, and Ü. Özgüner, "DGPS/INS integrated positioning for control of automated vehicle," in *IEEE Conference on Intelligent Transportation System (ITSC2001)*, (Oakland, CA), pp. 172–178, Aug. 2001.

[76] Z. Xiang and Ü. Özgüner, "A 3D positioning system for off-road autonomous vehicles," in *Proceedings of IEEE Intelligent Vehicles Symposium*, (Las Vegas, NV), pp. 130–135, June 2005.

[77] K. Watanabe and K. Izumi, "A survey of robotic control systems constructed by using evolutionary computions," in *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics (SMC'99)*, pp. 758–763, Oct. 1999.

[78] S. Kambhampati and L. Davis, "Multiresolution path planning for mobile robots," *IEEE Journal on Robotics and Automation*, vol. 2, pp. 135–145, Sept. 1986.

[79] Y. Hwang and N. Ahuja, "Path planning using a potential field representation," in *Proc. IEEE International Conference on Robotics and Automation*, (Philadelphia, PA), pp. 648–649, Apr. 1988.

[80] C. Warren, "Global path planning using artificial potential fields," in *Proc. IEEE International Conference on Robotics and Automation*, (Scottsdale, AZ), pp. 316–321, May 1989.

[81] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Proc. IEEE International Conference on Robotics and Automation*, (St. Louis, MO), pp. 500–505, Mar. 1985.

[82] J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 19, pp. 1179–1187, Sept.-Oct. 1989.

[83] M. Adams and P. Probert, "Towards a real-tme navigation strategy for a mobile robot," in *IEEE International Workshop on Intelligent Robots and Systems IROS'90*, (Tsuchiura,Japan), pp. 743–748, July 1990.

[84] L. A. Zadeh, "Fuzzy sets," *Information Control*, vol. 8, pp. 338–353, 1965.

[85] A. Saffiotti, "The uses of fuzzy logic in autonomous robot navigation," *Soft Computing*, vol. 1, pp. 180–197, Dec. 1997.

[86] T. Takeuchi, Y. Nagai, and N. Enomoto, "Fuzzy control of a mobile robot for obstacle avoidance," *Information Science*, vol. 45, pp. 231–248, 1998.

[87] H. Seraji, "Fuzzy traversability index: A new concept for terrain-based navigation," *Journal of Robotic Systems*, vol. 17, no. 2, pp. 75–91, 2000.

[88] T. Lee, L. Lai, and C. Wu, "A fuzzy algorithm for navigation of mobile robots in unknown environments," in *IEEE International Symposium on Circuits and Systems, 2005 (ISCAS 2005).*, (Kobe, Japan), pp. 3039–3042, May 2005.

[89] A. Davies and P. Samuels, *An Introduction to Computational Geometry for Curves and Surfaces*. Oxford: Clarendon Press, 1996.

[90] A. Broggi, M. Bertozzi, A. Fascioli, C. Guarino Lo Bianco, and A. Piazzi, "The ARGO autonomous vehicle's vision and control systems," *International Journal of Intelligent Ctonrol Systems*, pp. 409–441, 1999.

[91] J.-P. Laumond, ed., *Robot Motion Planning and Control*. Springer, 1998.

[92] J. A. Reeds and L. A. Shepp, "Optimal paths for a car that goes both forwards and backwards," *Pacific Journal of Mathematics*, 1990.

[93] N. Leonard and P. Krishnaprasad, "Motion control of drift-free, left-invariant systems on lie groups," *IEEE Transactions on Automatic Control*, pp. 1539–1554, Sept. 1995.

[94] A. Anzaldo-Menesesl, F. Monroy-Perez, B. Bonnard, and J. Gauthier, eds., *Contemporary trends in nonlinear geometric control theory and its applications*. Singapore ; River Edge, NJ : World Scientific, 2002.

[95] H. Sussmann and G. Tang, "Shortest paths for the reeds-shepp car: a worked out example of the use of geometric techniques in nonlinear optimal control," *Research Report SYCON-91-10, Rutgers University, New Brunswick, NJ*, 1991.

[96] J.-D. Boissonnat, A. Cerezo, and J. Leblond, "Shortest paths of bounded curvature in the plane," in *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 2315–2320, May 1992.

[97] D. P. Bertsekas, *Dynamic Programming and Optimal Control*. Belmont, Massachusetts: Athena Scientific, 2000.

[98] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, (Second Edition)*. MIT Press and McGraw-Hill. Section 24.3: Dijkstra's algorithm.