有全方位監控之多輛電腦視覺自動車的最佳安全巡邏之研究

研究生：陳倖嘉　　　　　指導教授：蔡文祥　教授

國立交通大學多媒體工程研究所

摘要

本論文提出利用多輛視覺型自動車執行安全巡邏工作的系統。我們假設巡邏環境的地板形狀是由矩形區塊所組成，並採用兩輛裝設有攝影機的自動車，以及兩部固定於天花板的魚眼鏡頭攝影機做為實驗平台。我們提出了一項獲取未知環境資訊的方法，可得到組成環境的所有矩形區域、導航自動車的轉折點、每組監控點間的距離與巡邏路徑。利用這些資訊可使自動車在航行時，不會與牆壁產生擦撞。另外，針對相機校正的問題，我們亦提出一項結合影像內插的點對應方法，利用多組二維影像與三維真實空間的對應點與影像內插法，去求得點在扭曲影像中的實際正確位置。此外，為了提高相機校正的精準度，我們也提出一項快速求取多組對應點的方法。藉由這些方法，我們可利用固定於天花板上的俯視攝影機來學習自動車相對於監控物品的位置與朝向，並利用此資訊來執行安全巡邏的工作。再者，我們也用俯視攝影機來定位與監控自動車的移動行為。而為了使得本系統有較好效益，我們則提出一項具有優化隨機與工作量平衡特性的路徑規劃方法，來分配巡邏路徑給各自動車，縮短完成每輪巡邏工作所花費的時間，進而提高環境中的安全程度。最後，由於本系統中所採用的自動車數量不只一台，所以我們提出了即時避碰的技巧，根據可能產生碰撞的路線狀況，自動產生可行駛的避碰路線。實驗結果證明我們所提出的這些方法是可行而且有效的。

A Study on Optimal Security Patrolling by Multiple Vision-Based Autonomous Vehicles with Omni-Monitoring

Student: Hsing-Chia Chen Advisor: Prof. Wen-Hsiang Tsai, Ph. D.

Institute of Multimedia Engineering, College of Computer Science

National Chiao Tung University

ABSTRACT

A multiple vision-based vehicle system for security patrolling in an indoor environment, whose floor shape is composed of rectangular regions, is proposed. Two autonomous vehicles controllable by wireless communication and equipped with cameras, as well as two cameras with fish-eye lenses fixed on the ceiling, are used as a test bed. To acquire information of an unknown environment, an environment-information calculation method is proposed for obtaining all rectangular regions composing the floor shape of the environment, the turning points for navigation, all distances between monitored objects, and the patrolling paths. These data enable the vehicles to navigate without collisions with walls. Also, a point-correspondence technique integrated with an image interpolation method is proposed for camera calibration. By a technique of finding corresponding points in 2-D image and 3-D global spaces as well as an image interpolation method, the correct positions of interesting feature points can be obtained from the warped images captured by the cameras with fish-eye lenses. Besides, a faster point-correspondence technique is proposed to obtain abundant corresponding points that yield better calibration accuracy. With this camera calibration technique, the cameras on the ceiling can be utilized to learn the poses of the vehicles with respect to monitored objects. Also, the vehicles are taught where and in which direction to perform the security monitoring task, in which the position information is used to guide the vehicles. Additionally, the top-view cameras can also be utilized to locate the vehicles and monitor vehicle activities in the navigation phase. An optimal randomized and load-balanced path planning method is proposed as well, which requires shorter time to accomplish object monitoring in one session and provides higher degrees of patrolling security. Because the number of the vehicles used in this study is more than one, a real-time collision avoidance technique is also proposed. According to the state of path-intersecting, feasible alternative paths for the vehicles can be obtained. Good experimental results show the flexibility and feasibility of the proposed methods for the application of multiple-vehicle security patrolling.

ACKNOWLEDGEMENTS

I am in hearty appreciation of the continuous guidance, discussions, support, and encouragement received from my advisor, Dr. Wen-Hsiang Tsai, not only in the development of this thesis, but also in every aspect of my personal growth.

Thanks are due to Mr. Chih-Jen Wu, Mr. Che-Wei Lee, Miss Shung-Yung Tsai, Mr. Guan-Lin Huang, and Mr. Jiun-Tsung Wang for their valuable discussions, suggestions, and encouragement. Appreciation is also given to the colleagues of the Computer Vision Laboratory in the Institute of Computer Science and Engineering at National Chiao Tung University for their suggestions and help during my thesis study.

Finally, I also extend my profound thanks to my family for their lasting love, care, and encouragement. I dedicate this dissertation to my beloved parents.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1
Introduction

1.1 Motivation of Study

Applications of intelligent robots are increasing gradually. An autonomous vacuum cleaner is a famous instance. The autonomous vehicle used in this research is also a robot, but it can only move. In order to add more ability to it, the vehicle is equipped with a camera. With the camera and its movement, the view of the vehicle is extended to a wider range. Such a kind of vision-based autonomous vehicle can perform more complicated tasks, such as security patrolling. It also can replace human beings to do dangerous or dreary works, for example, unknown object clipping, interoffice document delivering, etc.

A traditional security surveillance system is passive and restricted by its fixed position. The autonomous vehicle utilized to assist a security surveillance system can send an alert message to the security center actively when it detects an abnormal state. This provides more efficient and reliable security protection.

Security patrolling by *multiple* vision-based autonomous vehicles is more efficient than by one, because less time is taken to complete one session of patrolling all monitored objects. An additional advantage is that the shorter time interval between two objects patrolled increases the degree of security.

To have more benefits, a good planning of patrolling paths for all autonomous vehicles is important. Randomization, optimization, and load balancing are three critical principles that influence path planning. Autonomous vehicles patrolling randomly make thieves have no idea about when an object is not monitored by any vehicle. Optimal paths and load balances among all vehicles can decrease the time for all monitored objects to be patrolled once.

In order that an autonomous vehicle can carry out the patrolling task without any manpower, it has to be guided *smartly*. Ideas of learning artificial landmarks or specific scene features in the environment and locating the vehicle by landmark or feature matching have been developed intensively in the past decade. But most of them are restricted to be applicable in ideal environments, such as pure-colored backgrounds. Therefore, a top-view omni-camera with a fish-eye lens is utilized in this study to widen the applicable environment. The camera not only can locate autonomous vehicles but also can check ceaselessly whether they are still under control.

As a summary, our research goal in this study is to develop an autonomous vehicle security patrolling system with the following capabilities:

1. navigating automatically in environments whose floor shape is composed of rectangular regions;
2. monitoring and locating autonomous vehicles by top-view omni-cameras;
3. avoiding collisions between vehicles;
4. patrolling randomly; and
5. planning optimal paths and balanced loads for all autonomous vehicles.

1.2 Survey of Related Studies

In order to make autonomous vehicles navigate along a correct path, the vehicle location is the most vital information. Traditionally, an autonomous vehicle is equipped with an odometer to measure the current location of the vehicle. However, the vehicle usually suffers from incremental mechanic errors. Thus we need a technique of vehicle location estimation to correct the mechanic error in the navigation session.

For vehicle calibration, the geometric shapes of object boundaries [1, 2] or those labeled by users are utilized frequently [3, 4]. Furthermore, natural landmarks, such as house corners [5, 6], and the SIFT features of images [7] are also used in the techniques of vehicle calibration. In recent years, techniques of integrating laser range finders with conventional imaging devices have been proposed [8, 9].

In this study, a top-view omni-camera with a fish-eye lens is utilized to locate an autonomous vehicle. The camera must be calibrated before being used. Traditionally, we must calculate intrinsic and extrinsic parameters of the camera in order to obtain a projection matrix for transforming points between 2-D image and 3-D global spaces. A point-correspondence technique integrated with an image interpolation method is proposed, which is inspired by a technique coming from Lai and Tsai [10]. Because the camera is equipped with a fish-eye lens, images captured by it are warped. Winters and Santo-Victor [11] proposed a method for calibrating warped panoramic images. However, we can obtain a correct coordinate point directly by the camera calibration technique proposed by us.

Path planning is an important topic for the security patrolling by multiple vehicles. Many methods for this aim have been proposed in [12, 13, 14]. Besides, load balancing among all vehicles also need to be paid attention. Hert and Richards [15] proposed a method of using a polygon partitioning algorithm to achieve this objective. In this study, we propose a technique of calculating optimal and load-balanced paths in terms of some guidance points where vehicles perform security monitoring tasks. The idea of using guidance points comes from a learning method proposed by Chen and Tsai [16]. While vehicles navigate, collisions between vehicles must be avoided. Some methods [17, 18, 19] have been proposed to produce collision-free paths. To carry out optimal patrolling, we use the concept of the traveling salesman problem (TSP). Some methods for solving the TSP can be found in [20, 21, 22].

1.3 Overview of Proposed Approach

In this study, it is desired to develop a multiple vision-based autonomous vehicle system for security patrolling in an environment whose floor shape is composed of rectangular regions. In order to achieve this purpose, information about the environment, monitoring positions, and vehicle localization is quiet important. Therefore, some methods which can assist to acquire all of the above information are proposed and are roughly described in following. With such information, a technique which makes multiple vision-based autonomous vehicles navigate on correct paths without collisions and perform assigned security patrolling tasks is proposed. We divide the work conducted by the system into two phases: the *learning phase* and the *navigation phase*. They are illustrated in Figure 1.1 and Figure 1.2, respectively.

The *learning phase* consists of five steps to obtain the information about the environment, monitoring positions, and vehicle localization. The first step is *calibrating cameras*. In this study, there is no need to calculate all parameters of a camera. We propose another camera calibration technique which utilizes a pattern with some symbols labeled manually or with some natural landmarks, and obtains all corresponding points between 2-D image and 3-D global spaces. To obtain corresponding points faster, we propose additionally a technique of calculating the intersections of some quadratic curves, followed by using an interpolation method to obtain the global-space position of each point in a camera image.

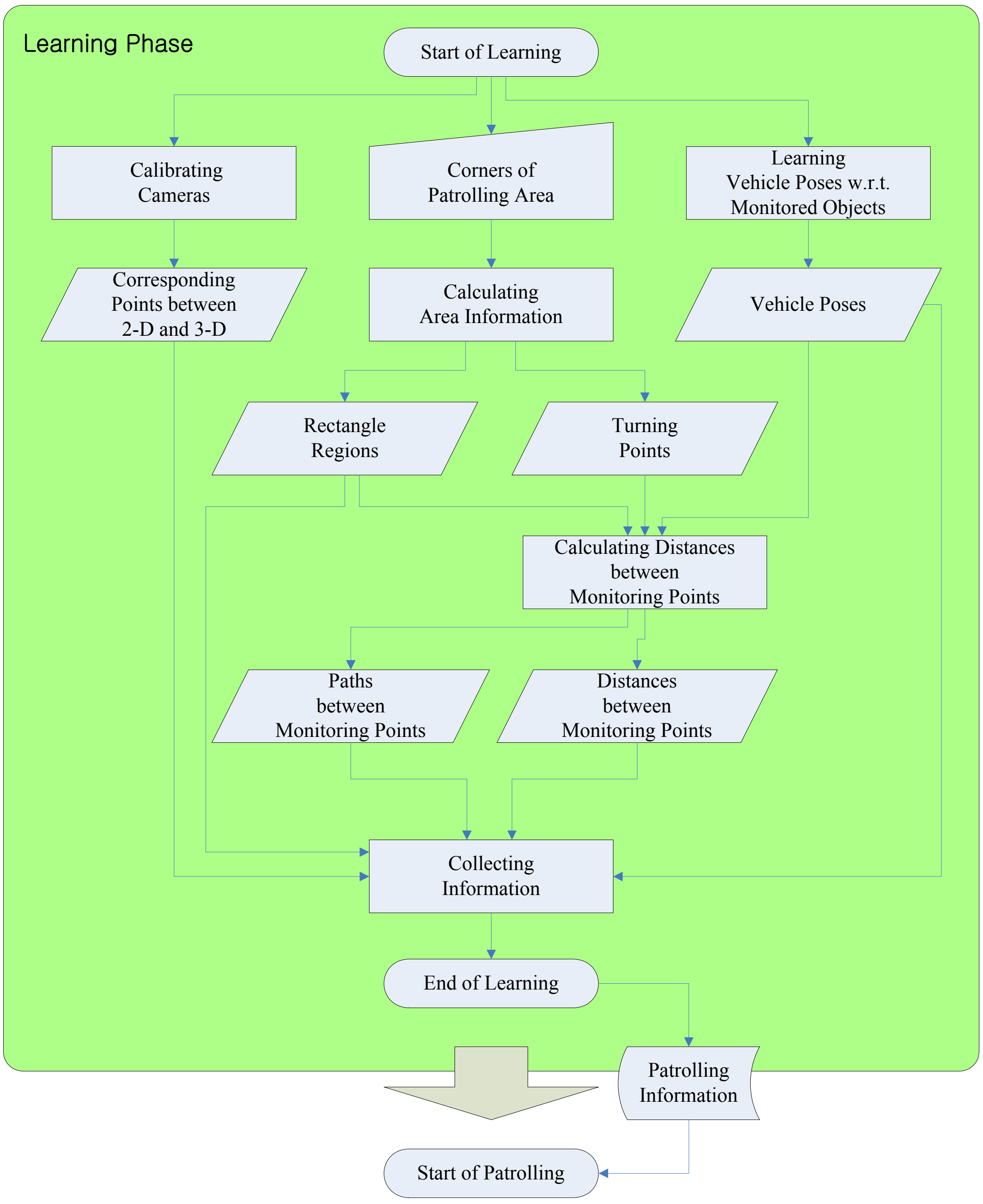

Learning Phase
Start of Learning
Calibrating Cameras
Corners of Patrolling Area
Learning Vehicle Poses w.r.t. Monitored Objects
Corresponding Points between 2-D and 3-D
Calculating Area Information
Vehicle Poses
Rectangle Regions
Turning Points
Calculating Distances between Monitoring Points
Paths between Monitoring Points
Distances between Monitoring Points
Collecting Information
End of Learning
Patrolling Information
Start of Patrolling

Figure 1.1 Flowchart of learning process of proposed system.

The second step is *calculating area information*, in which we take all corners of the patrolling area in the clockwise order as input, and find all rectangular regions and turning points by a method proposed in this study. The rectangular regions and turning points can prevent autonomous vehicles from colliding with walls in the environment.

The third step is *learning vehicle poses with respect to monitored objects*. While autonomous vehicles are patrolling, they must know where and in which direction to perform a security monitoring task. Therefore, an autonomous vehicle is driven to all positions where there are some monitored objects. Then, we point out the position of the autonomous vehicle in the image of a top-view omni-camera manually. We call the position a *monitoring point* in the sequel. For the direction, we utilize two positions of the vehicle to obtain a directional vector.

The fourth step is *calculating distance between each pair of monitoring points*. Because not all pairs of monitoring points are in the same rectangular region, an autonomous vehicle might not be able to navigate in a straight line between two monitoring points which are in different regions individually. Therefore, we propose a method for calculating the distance between any pair of monitoring points according to the information of rectangular regions, turning points, and positions of monitoring points obtained from the second and third steps of the learning phase described above. The distance is a critical factor that influences the decisions of patrolling paths. In addition, if two monitoring points belong to different regions, some turning points, which assist vehicles in moving from one monitoring point to the other without colliding with walls, are also recorded.

The final step of the learning phase is collecting information. All corresponding points between 2-D image and 3-D global spaces, rectangular regions, turning points, vehicle poses (including positions and directions), and navigational information between all pairs of monitoring points are collected to form a database. With all information in the database, autonomous vehicles will be able to perform the security patrolling task successfully.

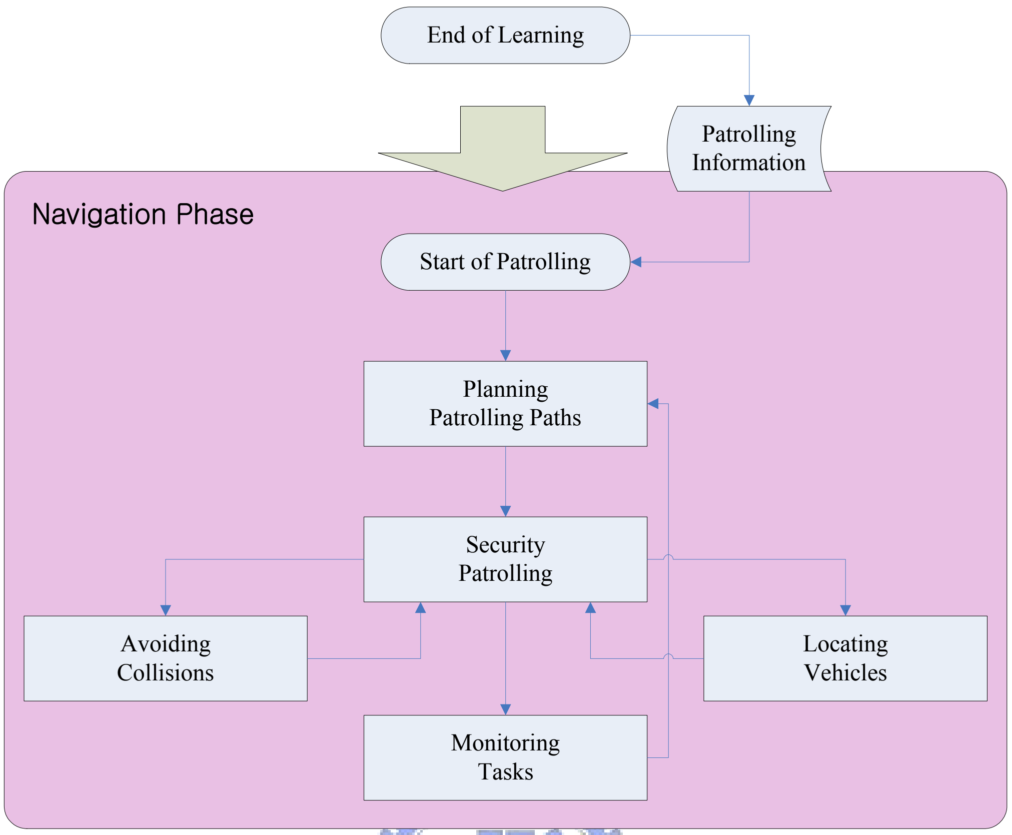


Figure 1.2 Flowchart of navigation of proposed system.

In the *navigation phase*, at first patrolling paths for all autonomous vehicles are decided according to the data obtained from the learning phase. For path planning, we propose a method for calculating optimal paths randomly with each monitoring point on these paths appearing only once. It means that all monitoring points are reached once *in one patrolling session*. With the path so planned, every autonomous vehicle can perform the security patrolling task at all monitoring points. When every autonomous vehicle completes traversing the assigned path, new patrolling paths are decided again.

Because of the existence of the mechanic errors of autonomous vehicles, it is important to locate all the autonomous vehicles in every session of patrolling. In this study, we propose a method of using a top-view omni-camera to do this job. Because the camera is fixed on the ceiling, we can obtain the absolute positions of vehicles from the images captured by the camera. When the position of an autonomous vehicle is obtained, the odometer value and direction angle of the vehicle can be corrected. The camera can also monitor whether autonomous vehicles are still under control. If not, the system will send an alert message to the security center and stop all vehicles.

In this study, two autonomous vehicles are utilized for conducting experiments of security patrolling. Hence, there may be collisions between them in patrolling sessions. To solve this problem, the odometer values of the vehicles are utilized to obtain the information about whether they are too close. The reason why the values of odometers are adopted is that these values may be corrected constantly by the top-view omni-cameras. As soon as the distance between the two vehicles is too close, a technique proposed in the study can make a change in paths to avoid the collision at once.

In summary, multiple vision-based autonomous vehicles can carry out security patrolling in environments whose floor shapes are composed of rectangular regions without collisions. By the top-view omni-cameras, autonomous vehicles can navigate along correct paths without collisions and whether the vehicles are still navigating normally also can be monitored. The patrolling paths are planned to be optimal and random. The loads of all autonomous vehicles are balanced. All of the above proposed techniques will bring a lot of merits for the application under investigation, namely, security patrolling by multiple autonomous vehicles.

1.4 Contributions

The main contributions of this study are summarized in the following.

(1) An environment-information aqusition method for collision avoidance between

the vehicles and walls is proposed.

(2) A point-correspondence technique integrated with an image interpolation method for camera calibration is proposed.

(3) A faster point-correspondence technique is proposed.

(4) A vehicle-pose learning method for performing the security monitoring task, which is to take pictures of monitored objects as defined in this study, and guiding the vehicles is proposed.

(5) An optimal method for randomized and load-balanced path planning is proposed.

(6) A vehicle localization and monitoring method by the top-view omni-cameras is proposed.

(7) A real-time collision avoidance technique between two vehicles is proposed.

1.5 Thesis Organization

The remainder of this thesis is organized as follows. In Chapter 2, we describe the system configuration of the vehicles used as a test bed in this study, as well as the principle of vehicle learning and guidance. In Chapter 3, the proposed techniques for camera calibration, acquiring information about environments, and patrolling tasks are described. In Chapter 4, the proposed methods for performing security patrolling and using top-view omni-cameras to localize and monitor vehicles are described. In Chapter 5, the proposed method for planning paths that are optimal, random, and load-balanced for all autonomous vehicles is described. In Chapter 6, the proposed method for collision avoidance between vehicles in patrolling sessions is described. Some experimental results are shown in Chapter 7. Finally, some conclusions and suggestions for future works are given in Chapter 8.

Chapter 2
System Configuration and Navigation Principles

2.1 Introduction

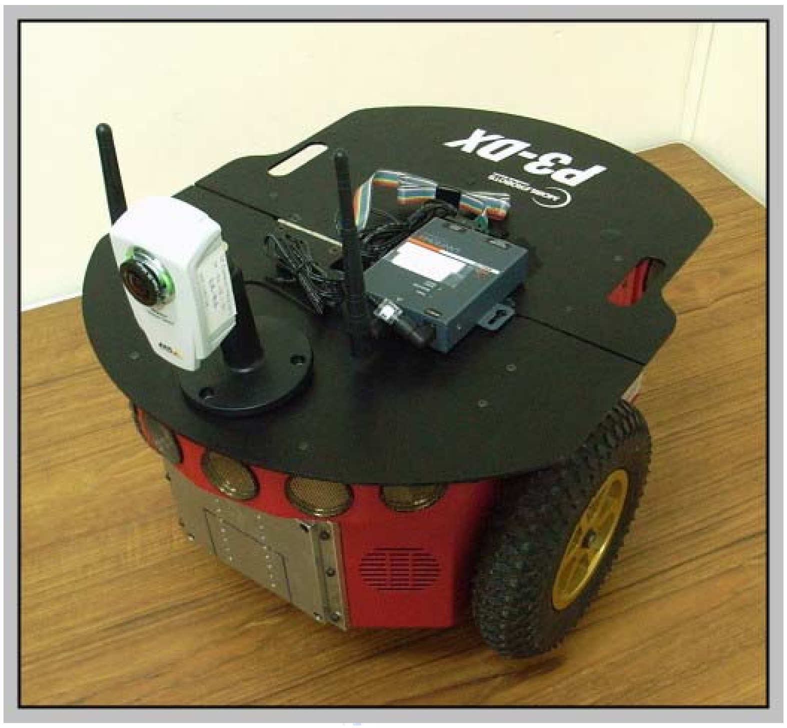
For security surveillance, the utilization of a vision-based autonomous vehicle is good for saving manpower. The vehicle is dexterous, with its moving ability increasing the view range of security surveillance. Besides, it can also monitor lower or hidden objects that may be under a table or in a cabinet.

In this study, two autonomous vehicles are used to perform the security patrolling task and each of them is equipped with a camera, as shown in Figure 2.1, though the proposed methods are general for any number of vehicles. Because the autonomous vehicles suffer from accumulation of mechanical errors, two cameras with fish-eye lenses, called *top-view omni-cameras* in the sequel, are installed on the ceiling. By the two cameras, autonomous vehicles can be located and controlled to navigate along correct paths. Between all on-board equipments and the user, some control and communication tools are required. The entire system configuration including hardware equipment and software are described in Section 2.2.

Before all autonomous vehicles carry out the security patrolling task, a *learning stage* is necessary, in which the vehicles are taught where to go, what to do, and how to avoid collision with walls. The process to obtain all information that makes autonomous vehicles be able to accomplish the task assignment is described in

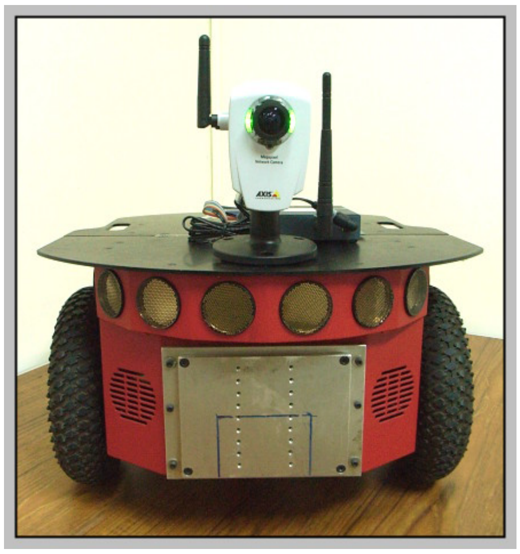
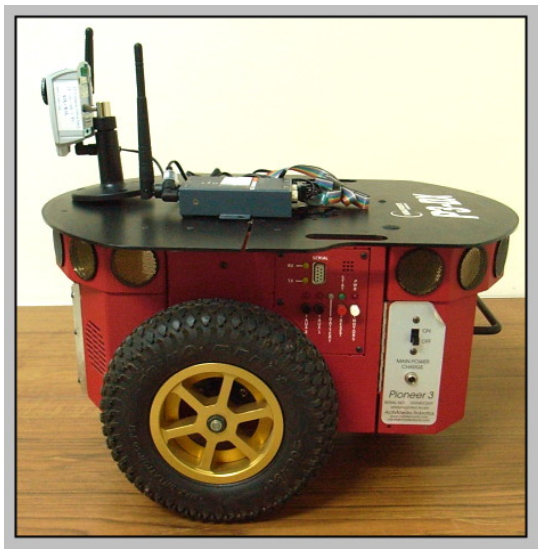
Section 2.3.

The phase in which autonomous vehicles carry out security patrolling is called the *navigation phase* in this study. In Section 2.4, we will describe the vehicle guidance principle and the process of performing the monitoring task in the navigation phase.



(a)

Figure 2.1 The vehicle used in this study is equipped with a camera. (a) A perspective view of the vehicle. (b) A front view of the vehicle. (c) A side view of the vehicle.

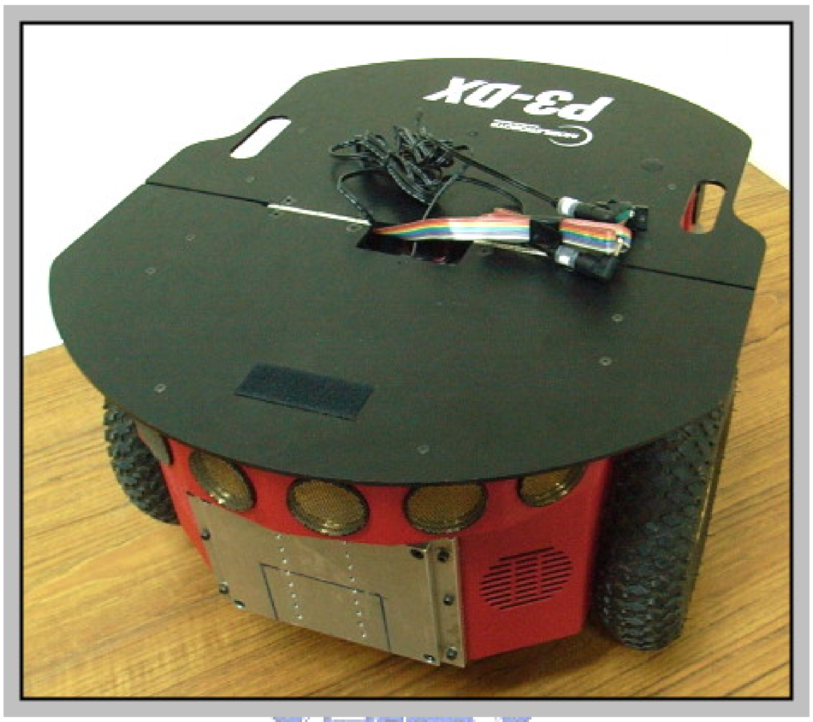
(b) (c)

Figure 2.1 The vehicle used in this study is equipped with a camera. (a) A perspective view of the vehicle. (b) A front view of the vehicle. (c) A side view of the vehicle. (continued)

2.2 System Configuration

In this study, the *vehicle system* used as a test bed is composed of a Pioneer 3-DX vehicle made by MobileRobots Inc., a WiBox made by Lantronix, and an Axis 207MW camera made by AXIS, as shown in Figure 2.2. The Axis 207MW camera, called the *camera system*, not only is one part of the vehicle system but also plays an important role of monitoring and locating vehicles. Because the whole system, called the *control system*, is controlled by users remotely, some wireless communication equipment is necessary. All the details of the above equipments are described in Section 2.2.1.

In order to develop the desired security surveillance system, we also need software that provides some commands and control interfaces. Besides, we also provide an interface for users to control the vehicles and cameras. All the above utilities are described in Section 2.2.2.



(a)

(b) (c)

Figure 2.2 The vehicle system used in this study. (a) A Pioneer 3-DX vehicle. (b) A WiBox. (c) An Axis 207MW camera.

2.2.1 Hardware configuration

The entire structure of the vehicle system used in this study is shown in Figure 2.3. There are three principal parts: vehicle system, camera system, and control system.

In the vehicle system, the *Pioneer 3-DX vehicle* is a 44cm×38cm×22cm aluminum body with two 19cm wheels and a caster. It can reach a speed of 1.6 meters per second on flat floors, and climb grades of 25^{o} and sills of 2.5cm. At slower speeds it can carry payloads up to 23 kg. The payloads include additional batteries and all accessories. By three 12V rechargeable lead-acid batteries, the vehicle can run 18-24 hours if the batteries are fully charged initially. A control system embedded in the vehicle makes the user's commands able to control the vehicle to move forward or backward or to turn around. The system can also return some status parameters of the vehicle to the user.

To show the advantage of the mobile vehicle, a wireless connection between a user and the vehicle is necessary. A *WiBox* is used to communicate with the vehicle by RS-232, so the user has the ability of remotely controlling the vehicle over a network from anywhere.

In the camera system, an Axis 207MW camera has the dimension of 85×55×40mm (3.3"×2.2"×1.6"), not including the antenna, and the weight of 190g (0.42 lb), not including the power supply, as shown in Figure 2.4. The maximum resolution of images is up to 1280×1024 pixels. In our experiment, the resolution of 320×240 pixels is used by the camera fixed on the vehicle and that of 640×480 pixels is used by the one fixed on the ceiling. Both of their frame rates are up to 15 fps. By wireless networks (IEEE 802.11b and 802.11g), captured images can be transmitted to users at speeds up to 54 Mbit/s. Each camera used in this study is equipped with a fish-eye lens that will expend the field of view.

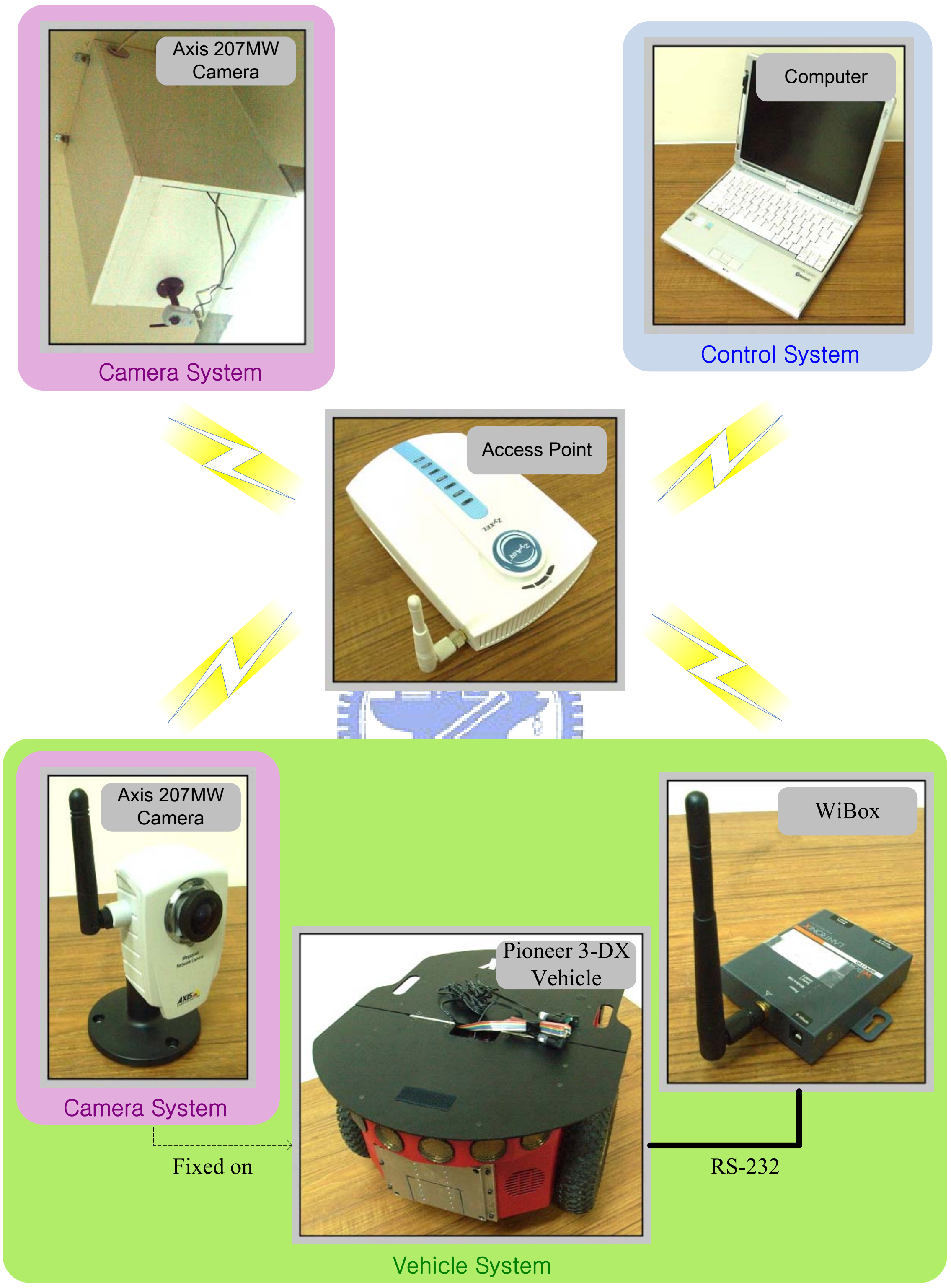
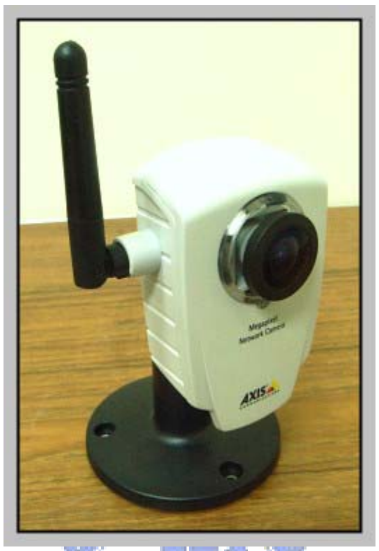


Figure 2.3 Structure of proposed system.

In the control system, a notebook PC is used to integrate the entire security patrolling system. With access points, all status information from vehicles and cameras can be delivered to the user by wireless networks. The PC produces some commands according to these data. By the same way, vehicles can receive the commands from the control system and perform corresponding actions. In other words, an access point is a communication medium among the three systems.



(a)

(b) (c)

Figure 2.4 The camera system used in this study. (a) A perspective view of the camera. (b) A front view of the camera. (c) A left-side view of the camera.

2.2.2 Software configuration

ARIA (Advanced Robotics Interface Application) provided by MobileRobots, Inc. is an API (application programming interface) that assists developers in communicating with the embedded system of the vehicle, either using a serial or TCP/IP connection. It is a powerful object-oriented toolkit and usable under Linux or Win32 OS in C++. Therefore, we use the Borland C++ builder as the development tool in our experiments to control the vehicles by ARIA. The lowest-level data and other information of the vehicle can also be retrieved easily by means of the ARIA interface.

About Axis 207MW camera controlling, the AXIS Company also provides a development tool called *AXIS Media Control SDK*. Using the Media Control ActiveX component from SDK, we can preview the image of the camera's view and capture the current image data. It is also convenient for users to use it to develop any function with the images grabbed from the camera as input.

2.3 Learning Principle and Proposed Process

Because the patrolling environment is unknown, a learning strategy is necessary. For the purpose of learning all knowledge that makes the vehicles accomplish the mission successfully, we develop a learning interface for users. The entire learning process is shown in Figure 2.5.

In this study, data having to be recorded are camera-related, object-related, and area-related ones. The *camera-related data* are obtained from a camera calibration process. In this study, we don't use the traditional camera calibration method to find a projective matrix for coordinate transformation. Instead, some landmarks on a pattern are utilized to acquire corresponding points between 2-D image and 3-D global spaces. For the camera fixed to ceilings in this system, the pattern is just the patrolling floor and the landmarks on it are just the corners of rectangular-shaped tiles. A user points some landmarks in the image by the user interface with a mouse, and corresponding points in the global space are calculated. Because each camera, used in our system, is equipped with a fish-eye lens, images captured by them are warped. Therefore, we use a *bilinear interpolation* method to translate coordinates in images into global space by these corresponding points.

The *object-related data* are used to teach vehicles where to go and which direction to face when they perform the patrolling task. We drive a vehicle to the position where the vehicle can observe the monitored object and then record it as a monitoring point (*MP*) according to the image of a top-view omni-camera. For the purpose of learning the direction with respect to the object, we control the vehicle to face the object and let it move forward for a short distance. By the two positions of the vehicle (nodes), the direction angle can be obtained.

The *area-related* data are about the environment where the vehicles patrol. An assumption made in this system is that the floor shape of the environment is composed of rectangular regions. At first, a user must key in corners in the clockwise order manually, and then all rectangular regions will be obtained. There might exist some pairs of MPs not belonging to an identical rectangular region, between which a vehicle cannot move straightly. Therefore, some points, called *turning points*, are necessary and they can be obtained by processing all the rectangular regions. With these turning points, the distances of all pairs of MPs can be calculated and which turning points between two MPs are passed by can also be recorded.

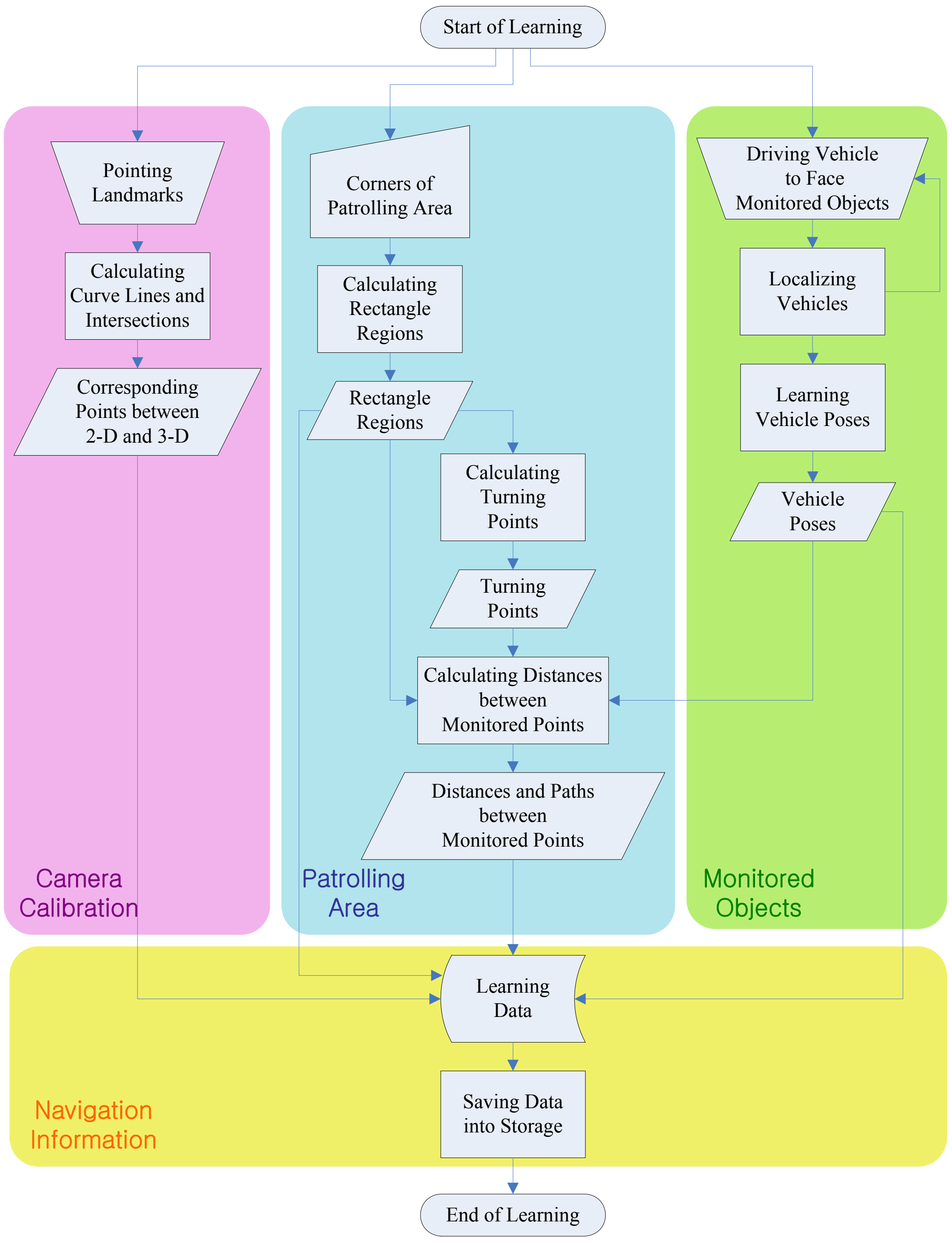

Start of Learning
Pointing Landmarks
Calculating Curve Lines and Intersections
Corresponding Points between 2-D and 3-D
Camera Calibration
Corners of Patrolling Area
Calculating Rectangle Regions
Rectangle Regions
Calculating Turning Points
Turning Points
Calculating Distances between Monitored Points
Distances and Paths between Monitored Points
Patrolling Area
Driving Vehicle to Face Monitored Objects
Localizing Vehicles
Learning Vehicle Poses
Vehicle Poses
Monitored Objects
Learning Data
Saving Data into Storage
Navigation Information
End of Learning

Figure 2.5 Flowchart of proposed learning process.

After all the data are obtained, they are saved into some text files. These files are then used in the navigation phase more than once.

2.4 Vehicle Guidance Principle and Proposed Process

When the learning job has been done, all vehicles can start to perform the security patrolling task. The entire guidance process proposed in this study is shown in Figure 2.6.

At first, the system reads all files that are obtained from the learning phase and contain information about the environment, autonomous vehicles, and monitored objects. According to the distances between all pairs of MPs, this system then plans random paths for each autonomous vehicle. If all differences between the paths of two vehicles do not exceed a threshold value which ensures the loads of all autonomous vehicles being balanced, the security patrolling task can be carried out.

Because autonomous vehicles suffer from accumulation of mechanical errors, we need to locate them constantly. When a vehicle runs a fixed length of distance, it must be located by the top-view omni-cameras. By the values of the vehicles' odometers, this system calculates the centroid of each vehicle from an image captured by a top-view omni-camera. The other function of the camera is to monitor vehicles to see whether they are still under control. If any vehicle loses control of its action, the system will stop all vehicles and send an alarm message to the user. Otherwise, the odometer of the vehicle is corrected and then the vehicle proceeds to move to its goal node.

While the vehicles are carrying out the security patrolling, there could be collisions between vehicles. Therefore, the detection of collisions is necessary. This system computes the distance between two vehicles in every cycle of a fixed time duration and determines if they are too close. If true, the paths of the vehicles will be changed.

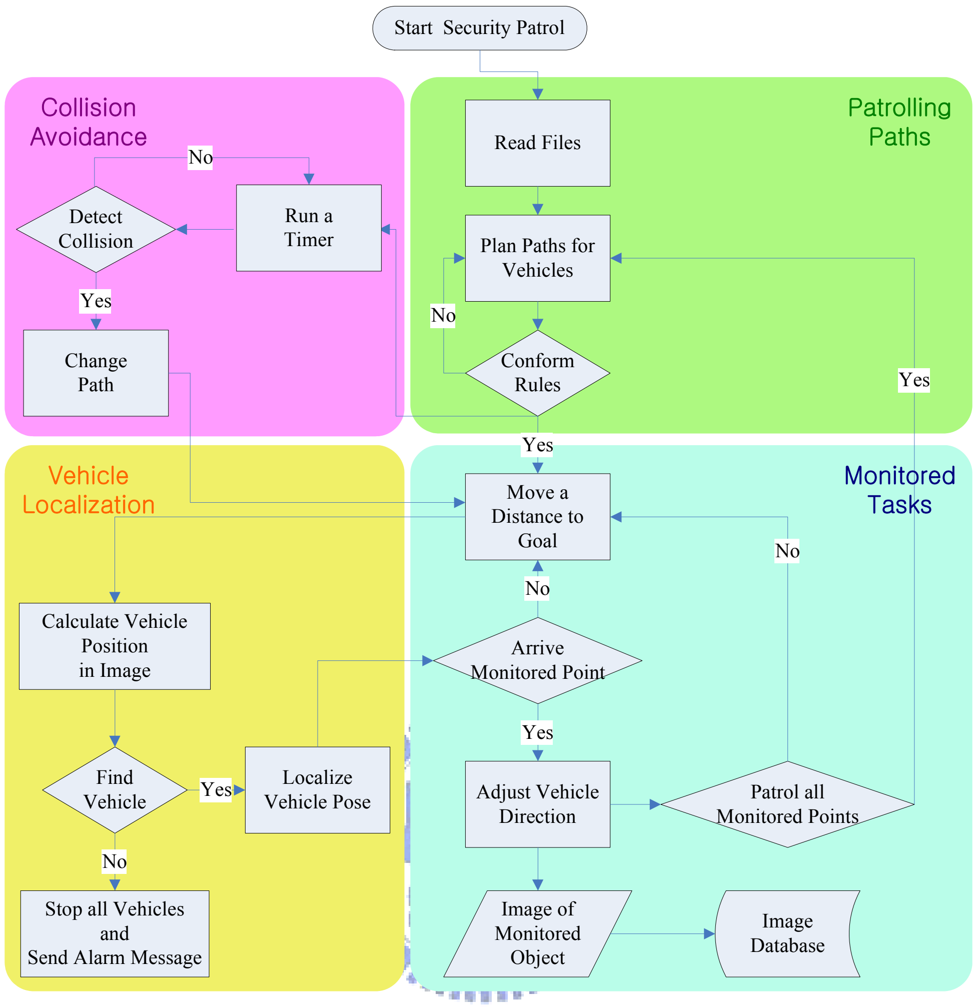


Figure 2.6 Flowchart of proposed navigation process.

A mission for the autonomous vehicles in this system is to take pictures of all the monitored objects during the navigation process. As a vehicle goes to a MP, it means that the vehicle will be in front of a monitored object. Therefore, the direction of the vehicle must be adjusted to face the object. Then the camera equipped on the vehicle takes a picture at the moment. The picture is transmitted to the control system by the wireless network and saved into an image file. When all the vehicles have accomplished their own patrolling paths, one cycle of security patrolling is finished. Then, the system will plan another set of new random paths for all the autonomous vehicles again.

Chapter 3
Learning Strategies for Navigation by Semi-automatic Driving

3.1 Ideas of Proposed Techniques Used in Learning

In this study, two cameras with fish-eye lenses fixed on ceilings are utilized to locate and monitor all autonomous vehicles. Before the use of the cameras, they must be calibrated. For this purpose, we propose in this study a point-correspondence technique integrated with an image interpolation method without conducting the conventional task of calculating the projection matrix for transforming points between 2-D image and 3-D global spaces. At first, by a mouse a user points out some landmarks in an image of a calibration target which is selected to be the tile pattern on the floor of our experimental environment. The landmarks we use in this study are the crossing points of the grid formed by the tile pattern. Such crossing points for use as corresponding points are abundant which yield better calibration accuracy in the proposed point-correspondence technique for camera calibration. The detail is described in Section 3.2.

In an environment where autonomous vehicles navigate, it is indispensable to use some turning points in the navigation path to ensure no collision between the vehicles and the walls. To compute the turning points, the corner points of the walkable area are first utilized to acquire all rectangular regions within the entire area. Each region is then represented by its upper-left and lower-right points. With these points, the system can judge whether the vehicles can move straightly between any pair of nodes where the vehicles visit (including the turning points). If two nodes belong to different regions, the vehicle will be guided to pass some turning points. In other words, the turning point is a medium that enables the vehicles to navigate between any pair of nodes without incurring collisions with the walls. Therefore, a turning point is selected to be the intersection of the centerlines of two overlapping regions or the center of the overlapping boundary of two adjacent ones. The details of the proposed techniques about processing rectangular regions and computing turning points are described in Section 3.3.1.

Additionally, to take the pictures of monitored objects by cameras equipped on the autonomous vehicles, all nodes and directions with respect to the objects must be recorded. In this study, a learning technique is proposed to guide vision-based vehicles to capture pictures at suitable spots and directions. Two top-view omni-cameras are used. The process is described in detail in Section 3.3.2.

3.2 Calibration of Top-View Omni-Cameras with Fish-Eye Lenses

Each camera used in this study is equipped with a fish-eye lens. All images captured by the camera are warped. So the traditional camera calibration method of obtaining a global-space point via a projection-based transformation cannot be utilized directly; the cameras must be calibrated by another method, as mentioned previously. For this, we propose a point-correspondence technique integrated with an image interpolation method. By the way, it is noted that the correct coordinates in the global space can be obtained from a warped image directly, as done in this study.

3.2.1 Review of Conventional Camera Calibration Technique

In general, a projection matrix is utilized in conventional methods to do the job of camera calibration. There are two kinds of parameters in the matrix, which must be calibrated, namely, the *intrinsic and* the *extrinsic parameters*. The *intrinsic parameters* do not depend on the position and orientation of a camera in the global space and include the focal length f, the image center point (u_0, v_0), the aspect ratio (S_x, S_y), and the skew error θ_s of the camera. Because the coordinate system of a camera and the global space may not be the same, the *extrinsic parameters* related to the rotation angle θ and the translation vector (t_x, t_y, t_z) of the camera must be calibrated.

Based on the intrinsic and extrinsic parameters, the relation between points in 2-D image and 3-D global spaces may be described by Eq. (3.1) below [23], where the point $(u, v)^T$ is in the image coordinate system and the point $(x, y, z)^T$ is in the global coordinate system:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} \text{transformation} \\ \text{matrix of} \\ \text{intrinsic parameters} \end{pmatrix} \begin{pmatrix} \text{transformation} \\ \text{matrix of} \\ \text{extrinsic parameters} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

$$\equiv \begin{pmatrix} S_x f & -S_x f \cot\theta_s & u_0 & 0 \\ 0 & S_y f / \sin\theta_s & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \cos\theta & -\sin\theta & 0 & t_x \\ \sin\theta & \cos\theta & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}. \qquad (3.1)$$

3.2.2 Proposed Calibration Technique

In the proposed calibration technique, we divide the patrolling area on the floor of the environment into multiple grids at first and all corners of them are called *reference points*. These points are the crossing points of the boundaries of the rectangular-shaped tiles on the floor. For every reference point, both of its coordinates in an image and in the global space must be recorded, describing a pair of corresponding points between the image and the global spaces.

In order to acquire more corresponding points faster, we calculate all quadratic curves in the image of the patrolling floor area, of which the intersections are exactly the desired reference points. Note that because the images are captured by the cameras equipped with fish-eye lenses, a straight line in the global space appears as a quadratic curve in the image. Therefore, the technique is feasible. A quadratic curve can be calculated by three points at least. This property is utilized to find all curves.

More specifically, we use a *minimum mean-square-error (MMSE)* method to calculate all the curves. Assume that a curve L is to be computed, which includes three parameters a, b, c. If points (x_1, y_1), (x_2, y_2), ..., (x_n, y_n) belong to the curve L, we may acquire n curves which can be represented as a matrix in the form of $A\vec{w} = \vec{b}$, as shown in Eq. (3.2) below:

$$\begin{aligned}&L: y = a + bx + cx^2, (x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) \in L\\&\Rightarrow A\vec{w} = \vec{b}\\&\text{where } A = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 \end{bmatrix}, \vec{w} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}, \vec{b} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}.\end{aligned} \tag{3.2}$$

The vector $\vec{w}$ can be computed by the MMSE criterion, that is,

$$\begin{aligned}&J(\vec{w}) = E[|\vec{b} - A\vec{w}|^2];\\&\hat{w} = \arg\min_{\vec{w}} J(\vec{w}).\end{aligned} \tag{3.3}$$

Through a series of simplifications from Eq. (3.3), we can acquire a result as

$$(A^T A)\hat{w} = A^T \vec{b}. \tag{3.4}$$

If all $x_1, x_2, \ldots, x_n$ are not equal, the matrix $A^T A$ is invertible and Eq. (3.4) can be solved to be:

$$\hat{w} = (A^T A)^{-1} A^T \vec{b}. \tag{3.5}$$

Therefore, the curve L may be computed to be

$$\begin{aligned}L: y &= a + bx + cx^2\\&= [1 \quad x \quad x^2]\hat{w}.\end{aligned} \tag{3.6}$$

Before calculating a curve, a user has to input the index of the curve and indicate that the curve is horizontal or vertical. As a curve is obtained, the pixels passed by the curve must record the index of the curve. The process of acquiring all quadratic curves is described as Algorithm 3.1 below. An example is shown in Figure 3.1 which is a result of acquiring some horizontal and vertical quadratic curves by the algorithm and each curve is calculated by four points on it.

Algorithm 3.1 *Calculating a quadratic curve*.

Input: An image *I*, the number *n* of points needed to calculate a curve, the index *k* of the curve, and being horizontal or vertical for the curve.

Output: Pixels passed by the quadratic curve.

Steps:

```
Step 1.   Point out $n$ points on the curve in the image $I$.
Step 2.   Calculate the curve by the MMSE criterion as described previously.
Step 3.   Record index $k$ and the property of being horizontal or vertical into the
          table of pixels passed by the curve.
```

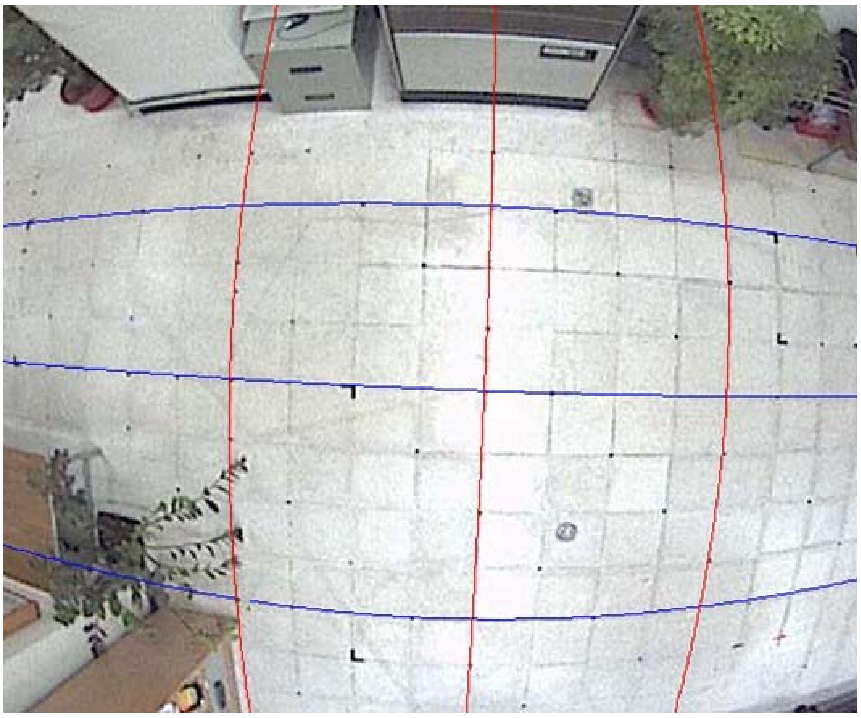


Figure 3.1 Calculating quadratic curves.

When all curves are obtained, we check all pixels in the image. If a pixel is passed both by a vertical curve and by a horizontal one, the pixel is taken to be one of the reference points. The width and height of a grid in the global space are also taken as input by the user. According to the lengths and the total numbers of horizontal and vertical curves, the coordinates of all reference points in the global space can be obtained. The entire process of acquiring all corresponding points between the image and global spaces is described as Algorithm 3.2 below. A result of finding all corresponding points from the image of one top-view omni-camera is shown in Figure 3.2.

Algorithm 3.2 *Acquiring corresponding points between 2-D image and 3-D global spaces*.

Input: An image *I*, the total number n_x of vertical curves, the total number n_y of horizontal curves, and the width w and height h of a grid in the global space.

Output: The coordinates of all corresponding points.

Steps:

Step 1. Repeat Algorithm 3.1 until enough curves are obtained.

Step 2. Find all intersections from the result of Step 1 and record their coordinates in the image coordinate system.

Step 3. Calculate all coordinates, with respect to the intersections obtained from Step 2, in the global coordinate system according to n_x, n_y, w and h, and then record them.

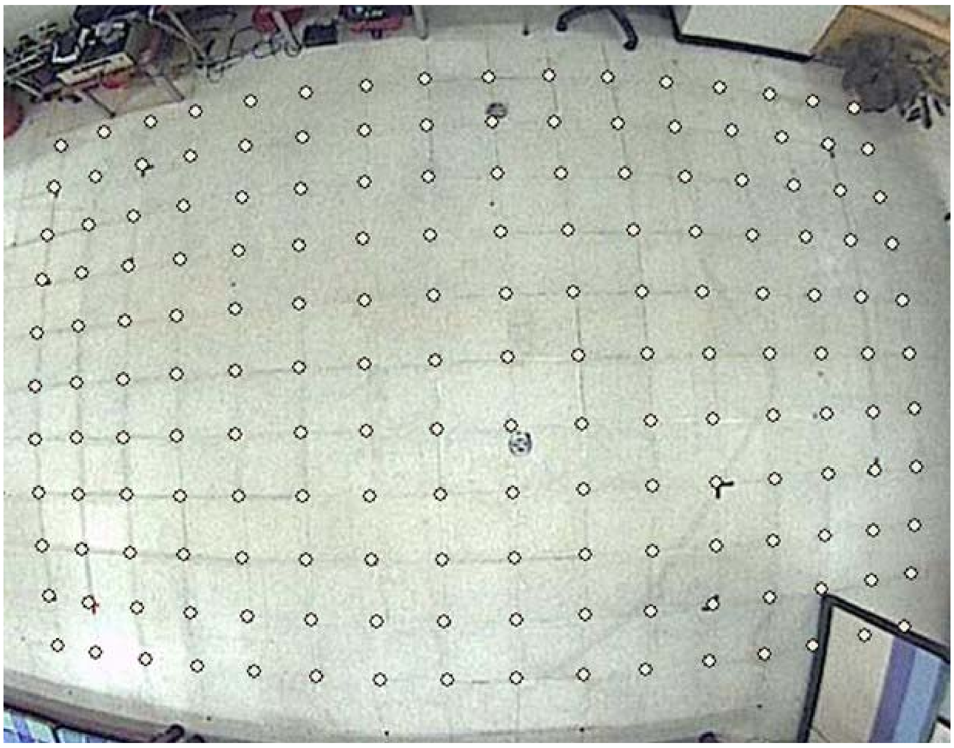


Figure 3.2 Finding all corresponding points.

With the corresponding points, a *bilinear interpolation* method is used in this study to obtain the global-space coordinates with respect to the points in the warped image. Some assumptions involved are shown in Figure 3.3. The points $P_{ij}, P_{(i+1)j}, P_{(i+1)(j+1)}, P_{i(j+1)}$ are the corners of the grid which point I belongs to. The lines L_0, L_1, L_2 and L_3 are straight lines through these corners. M_h is the line passing through point I and its slope is the mean of those of line L_1 and line L_3. M_v is the line passing through point I and its slope is the mean of those of line L_0 and line L_2. The points q and r are the intersections of M_h with L_0 and L_2, respectively. The points s and t are the intersections of M_v with L_1 and L_3, respectively.

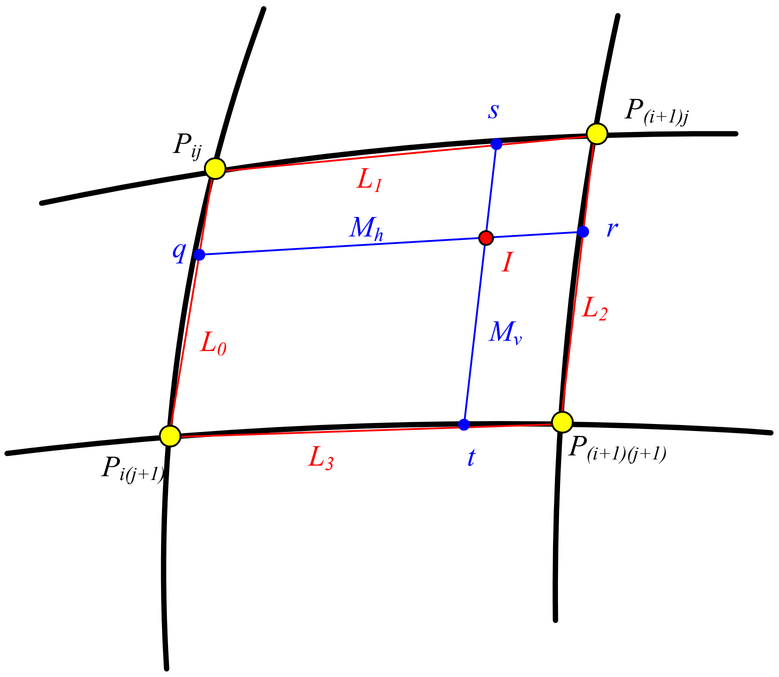
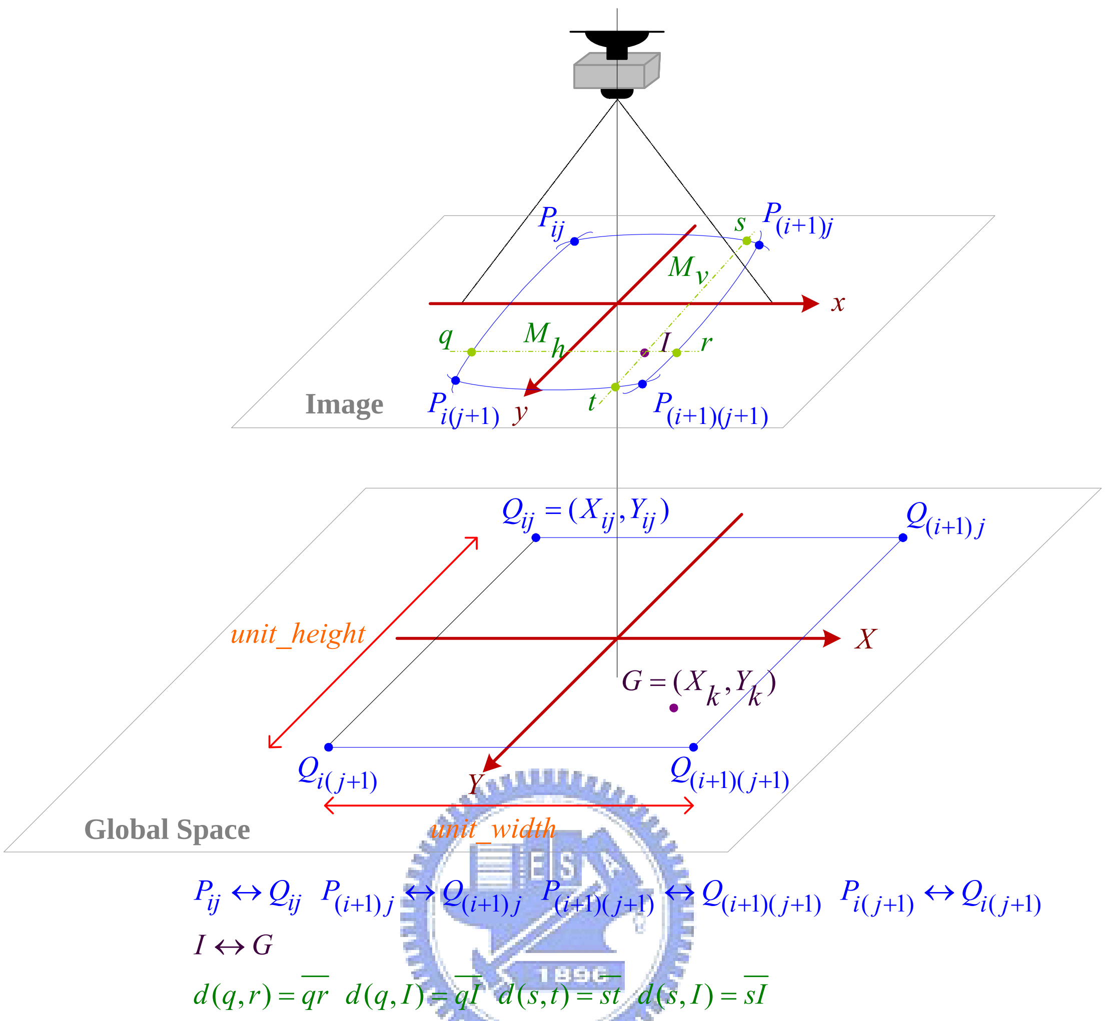


Figure 3.3 Point *I* in a distortion image.

For a point in a warped image, we must judge which grid the point belongs to and calculate M_h, M_v, q, r, s and t. The ratios of distances are utilized to obtain the global-space coordinates with respect to the point. The equation is Eq. (3.7) below:

$$\begin{aligned} X_k &= X_{ij} + unit\_width \times \frac{d(q,I)}{d(q,r)} \\ Y_k &= Y_{ij} + unit\_height \times \frac{d(s,I)}{d(s,t)}. \end{aligned} \tag{3.7}$$

The points $G(X_k, Y_k)$ and $Q_{ij}(X_{ij}, Y_{ij})$ are in the global space with respect to point I and point P_{ij}, respectively. The lengths *unit_width* and *unit_height* are the widths and the heights of a grid in the global space, respectively. The distance $d(q,I)$ is the length from point q to point I. A graphic illustration is shown in Figure 3.4 and the process of acquiring the coordinates in the global space is described as Algorithm 3.3.


P_{ij}
s
$P_{(i+1)j}$
M_v
x
q
M_h
I
r
Image
$P_{i(j+1)}$
y
t
$P_{(i+1)(j+1)}$
$Q_{ij} = (X_{ij}, Y_{ij})$
$Q_{(i+1)j}$
unit_height
X
$G = (X_k, Y_k)$
$Q_{i(j+1)}$
Y
$Q_{(i+1)(j+1)}$
Global Space
unit_width
$P_{ij} \leftrightarrow Q_{ij}$ $P_{(i+1)j} \leftrightarrow Q_{(i+1)j}$ $P_{(i+1)(j+1)} \leftrightarrow Q_{(i+1)(j+1)}$ $P_{i(j+1)} \leftrightarrow Q_{i(j+1)}$
$I \leftrightarrow G$
$d(q,r) = \overline{qr}$ $d(q,I) = \overline{qI}$ $d(s,t) = \overline{st}$ $d(s,I) = \overline{sI}$
Figure 3.4 A relation between the image taken by the omni-camera and the global space.

Algorithm 3.3 *Computation of coordinates in the global space.*

Input: A point I in a warped image, the coordinates of all corresponding points, the width *unit_width* and the height *unit_height* of a grid in the global space.

Output: The coordinates of the point G in global space with respect to the point I.

Steps:

Step 1. Judge which grid point I belongs to.

Step 2. Calculate lines L_0, L_1, L_2 and L_3 that are the straight lines through the corners of the grid to which point I belongs, as illustrated by Figure 3.3.

Step 3. Calculate line M_h that passes through point I with its slope being the mean of those of line L_1 and line L_3.

Step 4. Calculate line M_v that passes through point I with its slope being the mean of those of line L_0 and line L_2

Step 5. Calculate points q and r that are the intersections of line M_h and the edges of the grid.

Step 6. Calculate points s and t that are the intersections of line M_v and the edges of the grid.

Step 7. Calculate point $G(X_k, Y_k)$ by the following equation:

$$X_k = X_{ij} + unit\_width \times \frac{d(q,I)}{d(q,r)};$$

$$Y_k = Y_{ij} + unit\_height \times \frac{d(s,I)}{d(s,t)}.$$

Annotate that point $Q_{ij}(X_{ij}, Y_{ij})$ is in the global space with respect to point P_{ij} in the warped image.

3.3 Information for Security Patrolling

3.3.1 Proposed techniques of learning patrolling environment

Information about the patrolling environment includes rectangular regions, which the floor shape is composed of, and the turning points among the regions. To find all *rectangular regions*, a user must key in all the corners of the patrolling area in the clockwise order. Because each point in the area belongs to one rectangular region at least, a vertical line, called *scanning line*, is utilized to check whether some points in the vertical line do not belong to one region at least. If yes, another new rectangular region will be found for the points. The entire process and details are described as Algorithm 3.4 below.

Algorithm 3.4 *Finding all rectangular regions*.

Input: All corners of the patrolling area.

Output: All rectangular regions of the patrolling area.

Steps:

Step 1. Set a vertical line *L* through the leftmost corner of the patrolling area and scan the area from left to right.

Step 2. Find the intersections of *L* and the boundaries of the area, and record them. Record the corners of the boundary merely, if there is an overlap between *L* and the boundary.

See the example shown in the following, where points *a*, *b* and *c* recorded.

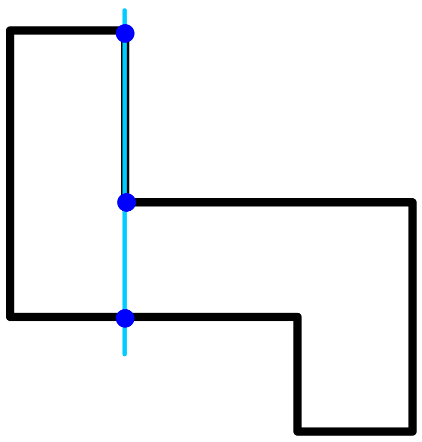


Figure 3.5 Intersections of L and the boundaries of the area.

Step 3. Divide these intersections into groups from top to down with each group including two points.

See the example of Step 2 above for an illustration, where two groups (*a*, *b*) and (*b*, *c*) are obtained.

Step 4. Judge where to set a *line segment l*.

Step 4.1. Check whether the right part of a group is the inner range of the patrolling area and a rectangular region in the range has not been found.

If yes, go to Step 4.2; else, continue to check the next group.

Annotate that the technique of checking whether the right part of a group is the inner range of the patrolling area is described in Algorithm 3.5.

Step 4.2. Set *l* in the following way.

Step 4.2.1. Shift the middle point *M* of the group one pixel to the right.

Step 4.2.2. Extend *M* along a vertical direction until colliding with the boundaries, where the line *l* is set.

See the example shown in the following, where line $\overline{de}$ is *l* mentioned above.

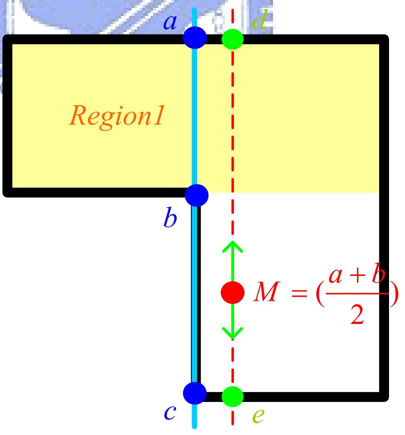


Figure 3.6 Setting a line segment *l*.

Step 5. Continue scanning the line segment *l* to the left until colliding with the boundary of the patrolling area.

See the example shown in the following.

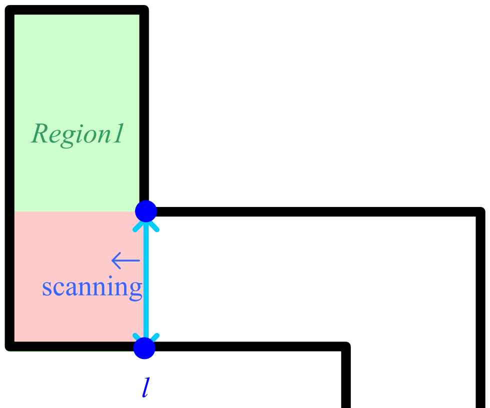


Figure 3.7 Scanning the line segment *l* to the left.

Step 6. Continue scanning the line segment *l* to the right until colliding with the boundary of the patrolling area.

See the example shown in Step 5, where *Region2* was obtained.

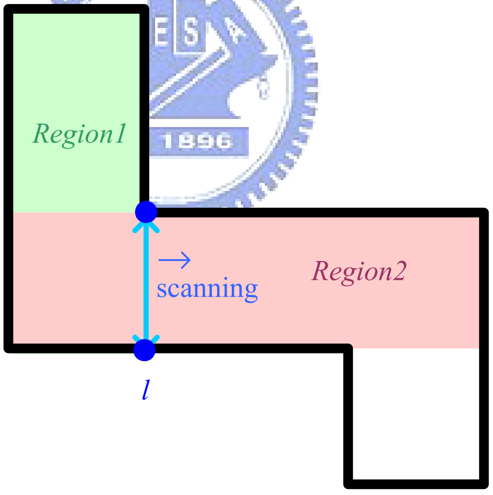


Figure 3.8 Scanning the line segment *l* to the right.

Step 7. Continue scanning the vertical line *L* to the right and repeat Step 2 until *L* reaches the rightmost boundary of the patrolling area.

Algorithm 3.5 *Judging an inner range.*

Input: All corners *C* of the patrolling area, and two points *P* whose *x*-coordinates are the same.

Output: An indication whether the right part of *P* is an inner range of the patrolling area.

Steps:

Step1. Calculate the middle point *M* of the two points *P*.

Step2. Shift *M* one pixel to the right.

Step3. Set a vertical line *LM* through point *M*.

Step4. Check the numbers of intersections of the line *LM* and the boundary.

If both the number of the upper part and that of the lower one of *M* are odd and either part does not include an overlapping line, it is decided that the right part of *P* is an inner range of the patrolling area.

See the graphic illustration shown in the following.

Case 1: the right part of points *a* and *b* is not an inner range of the patrolling area.

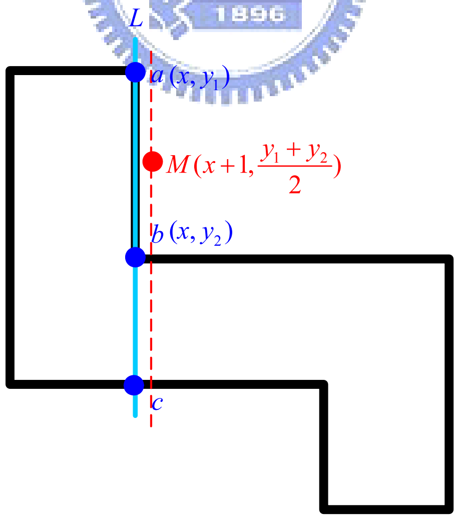


Figure 3.9 The right part of points *a* and *b* is not an inner range.

Case 2: the right part of points *b* and *c* is an inner range of the patrolling area.

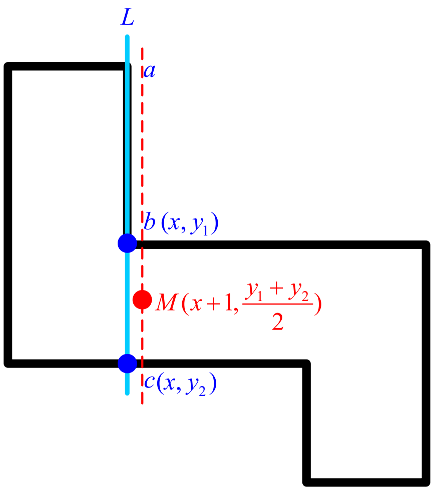


Figure 3.10 The right part of points *a* and *b* is an inner range.

Now, we show a result of Algorithm 3.4. A patrolling area is shown in Figure 3.11. The total number of rectangular regions in the patrolling area is seven and they are shown in Figure 3.12 through Figure 3.18 step by step.

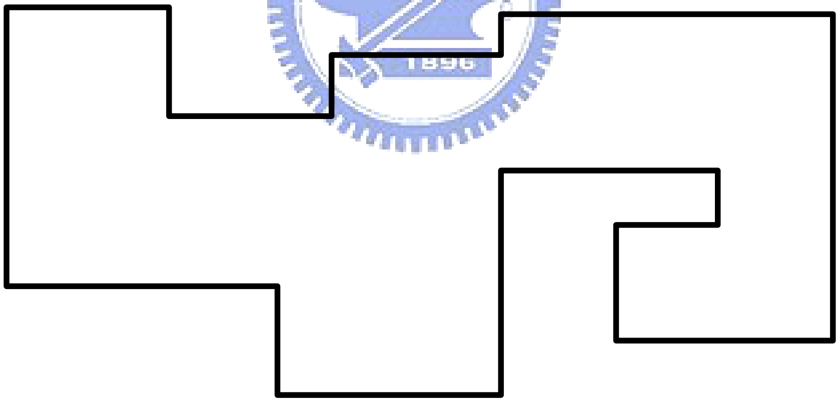


Figure 3.11 A patrolling area.

According to these rectangular regions, we can compute all desired turning points. If two regions are overlapping or adjacent, we can obtain a turning point. The point is the intersection of two vertical and horizontal centerlines of the regions. The process of finding all turning points is described as Algorithm 3.6 and the result in the patrolling area of Figure 3.11 is shown in Figure 3.19. The eight circles in Figure 3.19 are exactly the turning points, in which each blue one is the center of the overlapping boundary of two adjacent regions.

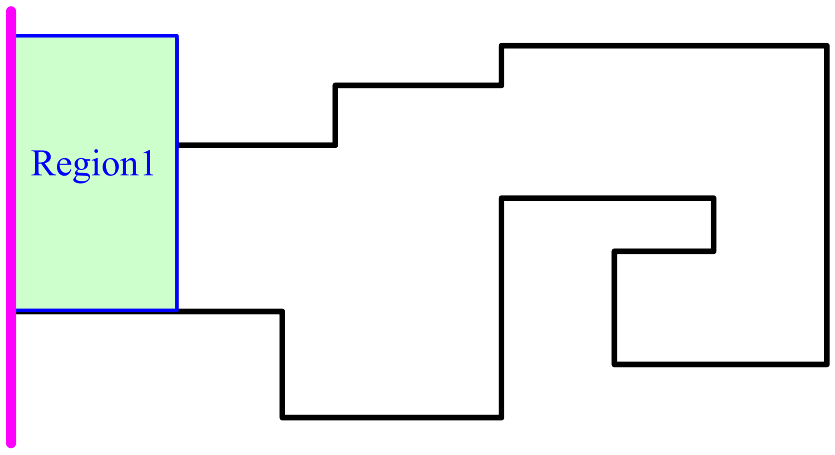


Figure 3.12 The first rectangular region.

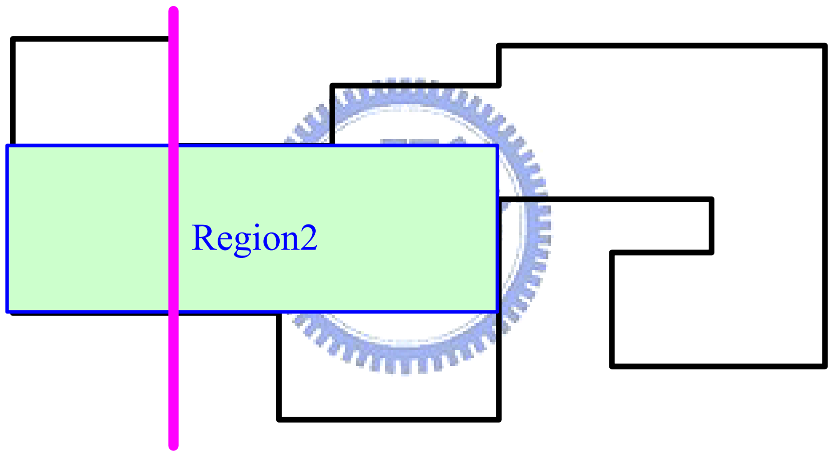


Figure 3.13 The second rectangular region.

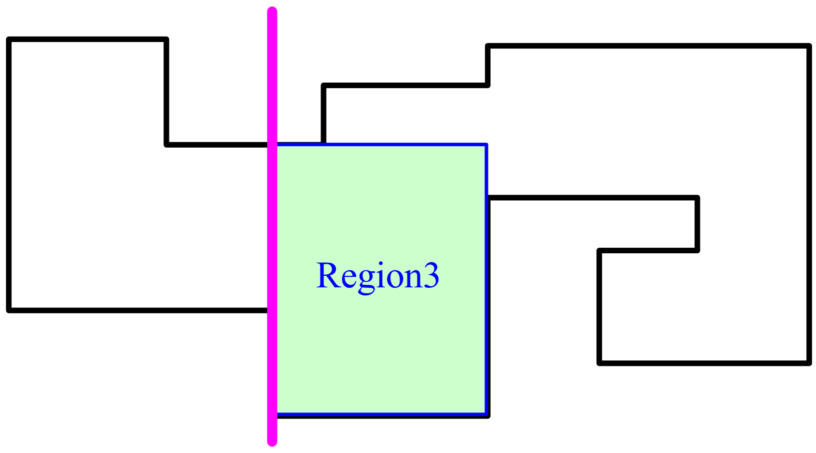


Figure 3.14 The third rectangular region.

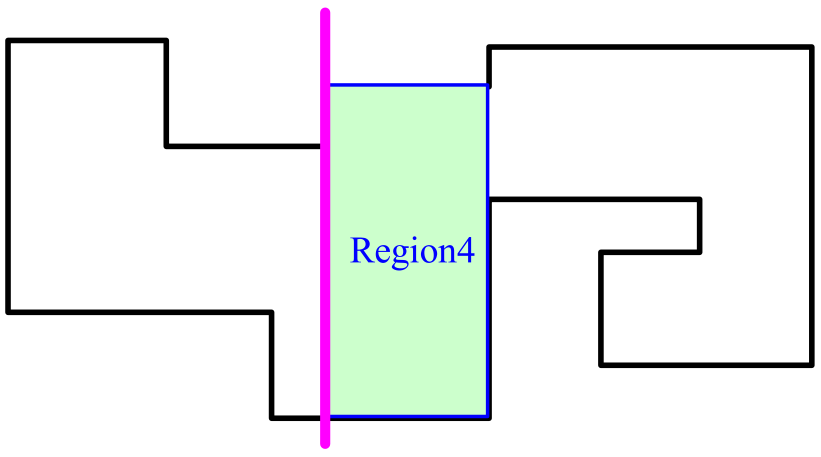


Figure 3.15 The fourth rectangular region.

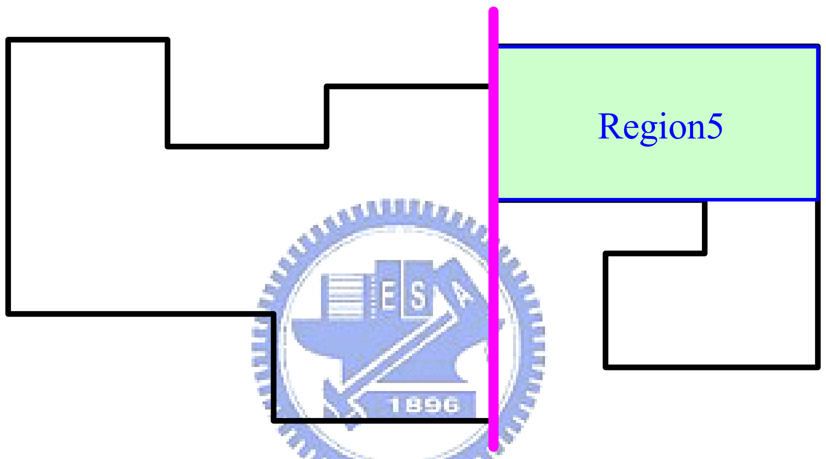


Figure 3.16 The fifth rectangular region.

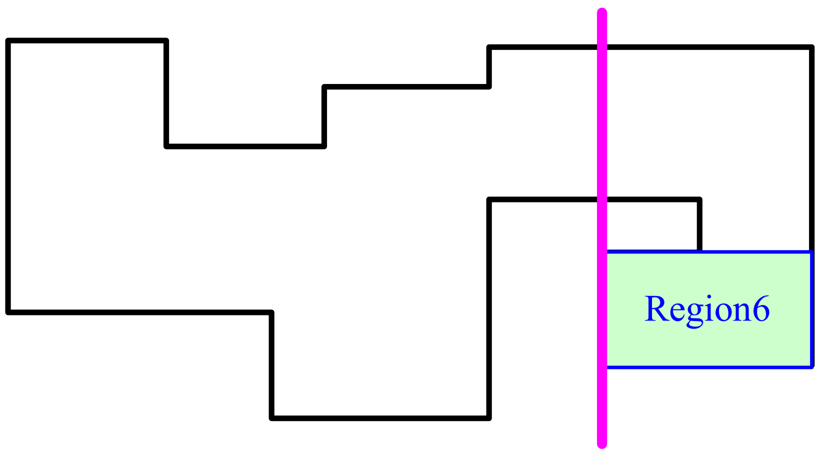


Figure 3.17 The sixth rectangular region.

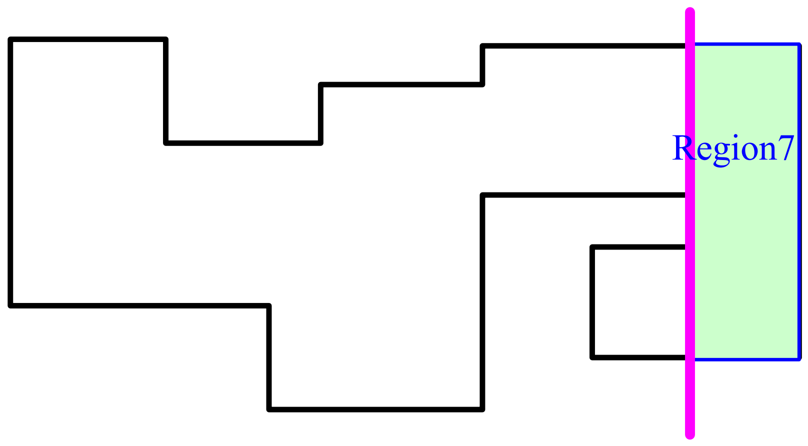


Figure 3.18 The seventh rectangular region.

Algorithm 3.6 *Finding all turning points*.

Input: All rectangular regions *R* in an area.

Output: All turning points.

Steps:

Step1. Take a pair of overlapping or adjacent regions from the set *R*.

If the two regions are overlapping, go to Step2; else, go to Step4.

Step2. Calculate all vertical and horizontal centerlines of the two regions.

Step3. Find an intersection within these centerlines, which is exactly a turning point of the area.

Step4. Calculate the center of overlapping boundary, which is exactly a turning point of the area.

Step5. Repeat Step 1 until all regions in *R* have been checked.

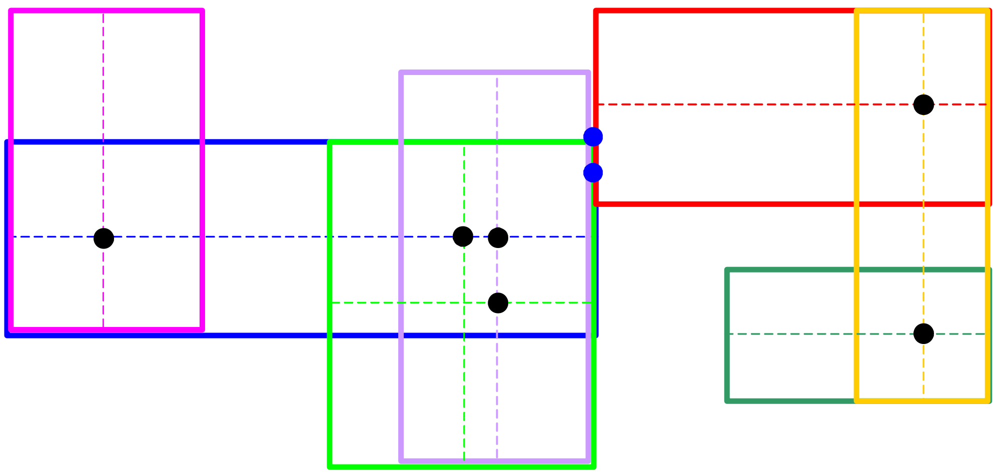


Figure 3.19 All turning points in the area.

3.3.2 Learning of poses of vehicles with respect to monitored objects

To learn the poses of autonomous vehicles with respect to monitored objects, we designed a user interface, as shown in Figure 3.20. The image in the interface is the real-time view of the camera installed on an autonomous vehicle.

Before learning, a vehicle is parked at the origin of the global space. Because the vehicle suffers from mechanic errors, a user must constantly locate it by the top-view omni-cameras in the period of the learning phase. We can use a joystick or the buttons in the user interface to control the moves of the vehicle. As the vehicle has moved a short distance, the user must press the "Localize" button in the interface. For an example, Figure 3.21 shows a situation that the vehicle is ready to be located. After the user presses the "Localize" button, the system will calculate the centroid of the vehicle in the image captured by a top-view omni-camera. Then, the centroid is transformed from the image space into the global space and the odometer value of the vehicle is corrected by the coordinates of the resulting point.

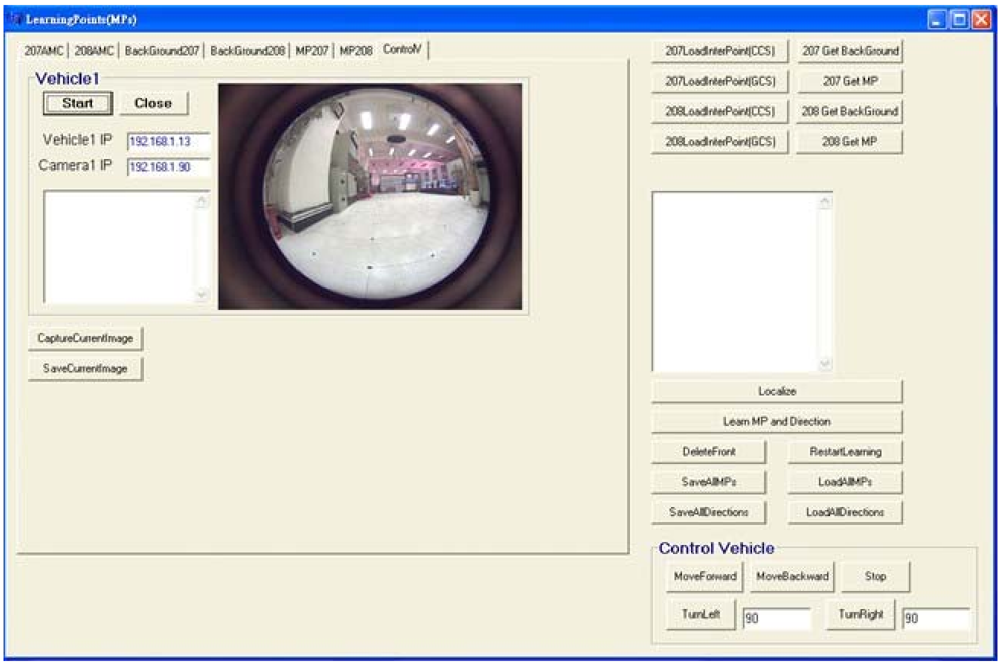
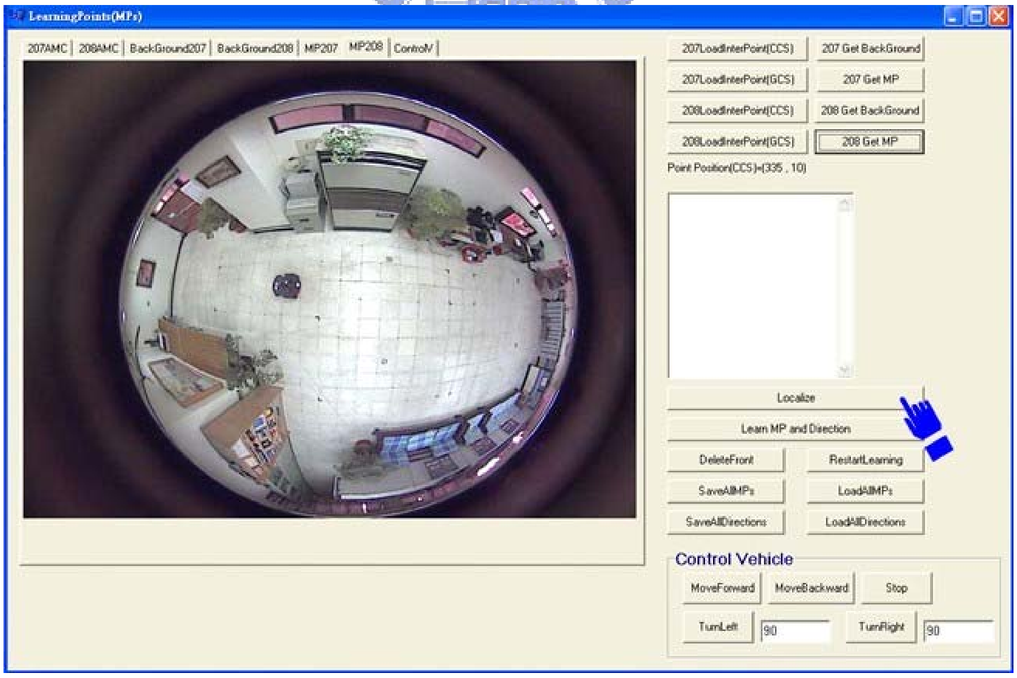


Figure 3.20 User interface of learning *MP*s.

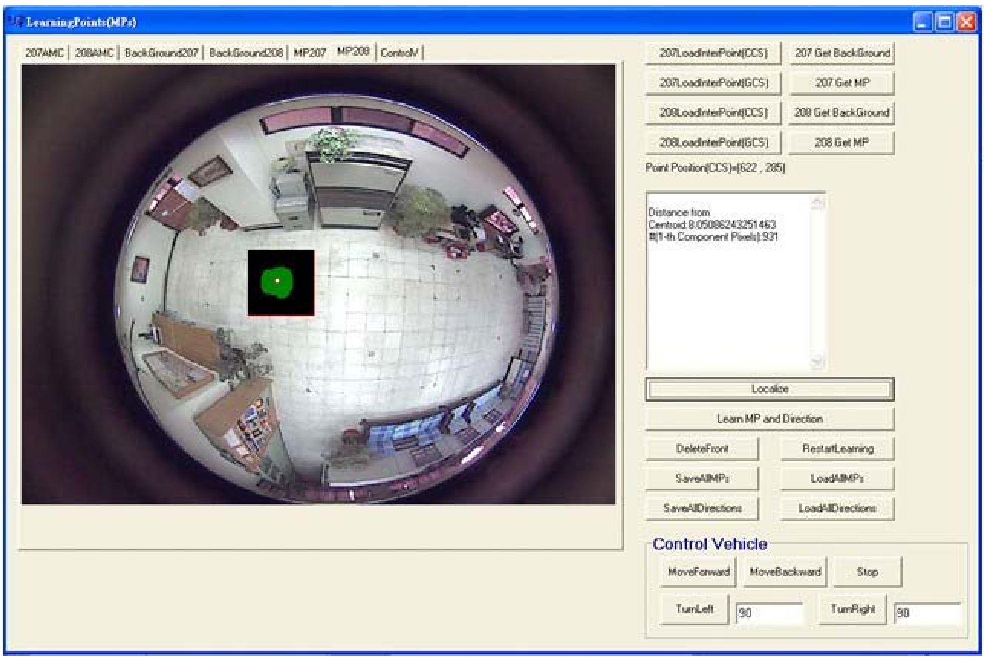


Point Position(CCS)=(335 , 10)

Figure 3.21 Before localizing the vehicle.

In Figure 3.22, the green component is the vehicle and a white circle in the component is its centroid. The entire detail of the technique is described in Section 4.2.2.

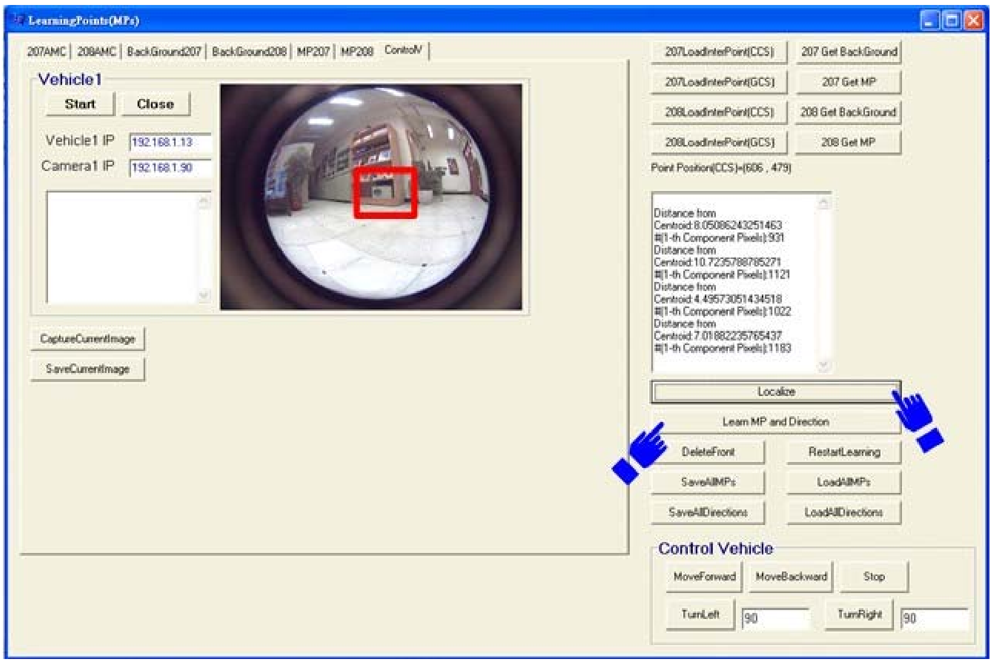
For the direction with respect to monitored objects, we utilize two points to obtain the direction vector. First, we drive the vehicle to the front of a monitored object, leaving a sufficient distance between the vehicle and the object. At the moment, we adjust the orientation of the vehicle by the image in the user interface. As the monitored object appears at the center of the image, we locate the vehicle by pressing the “Localize” button. Furthermore, we move the vehicle forward a short distance and press the “Learn MP and Direction” button. For the example shown in Figure 3.23, the safe deposit labeled by a red rectangle is a monitored object and the vehicle faces the object. The detail process is described as Algorithm 3.7.



```
Point Position(CCS)=(622 , 285)
```

```
Distance from
Centroid:8.05086243251463
#(1-th Component Pixels):931
```

Figure 3.22 Localizing the vehicle.



Vehicle1 IP 192.168.1.13

Camera1 IP 192.168.1.90

Point Position(CCS)=(606 , 479)

```
Distance from
Centroid:8.05086243251463
#(1-th Component Pixels):931
Distance from
Centroid:10.7235788785271
#(1-th Component Pixels):1121
Distance from
Centroid:4.49573051434518
#(1-th Component Pixels):1022
Distance from
Centroid:7.01882235765437
#(1-th Component Pixels):1183
```

TurnLeft 90

TurnRight 90

Figure 3.23 The vehicle in front of a monitored object.

Algorithm 3.7 *Learning poses of vehicles with respect to monitored objects*.

Input: The position *P* of the vehicle in front of a monitored object.

Output: The pose of the vehicle with respect to the object.

Steps:

Step 1. Drive the vehicle to the position *P*.

Step 2. Let the vehicle face the object.

Step 3. Localize the vehicle.

Step 4. Move the vehicle forward a short distance.

Step 5. Localize the vehicle again and save its pose.

Chapter 4
Security Patrolling by Multiple Vehicle Navigation

4.1 Introduction to Concepts in Proposed Systems

In the study, we use multiple vision-based autonomous vehicles to perform security patrolling. To obtain more benefits, there must be ideal path planning for all vehicles. By an optimal randomization technique proposed in this study and described in this chapter, these patrolling paths have the properties of randomization, optimization, and load balancing within all vehicles. Furthermore, two cameras with fish-eye lenses are utilized to localize and monitor all the vehicles. All concepts of the above are described in following five sections.

4.1.1 Randomized patrolling

The patrolling path of every autonomous vehicle is produced to be random. Random paths are good for security surveillance. Because a fixed path will reveal where a vehicle is located at a fixed moment, thieves can, by such observation, invade and steal those valuable objects which are not guarded by any vehicle at the time. To randomize a patrolling path, all nodes (*MP*s) in the path with respect to monitored objects are chosen randomly. Furthermore, if the time interval between two patrollings of a monitored object is smaller, the security of the objects will be raised. Therefore, each node (*MP*) is just chosen once in a patrolling session.

4.1.2 Optimal path

After the nodes of a patrolling path are determined, the order of passing these nodes is computed. To decrease the time taken to accomplish security patrolling in one patrolling session, the distance of each path must be made to be the shortest. A method for this purpose by finding the *Hamiltonian path* is utilized and the detail of determining all patrolling paths is described in Chapter 5.

4.1.3 Load balance

To obtain more benefits of using multiple autonomous vehicles to perform the security patrolling, the loads for all vehicles must be balanced. We set a *threshold parameter* to restrict the differences of the patrolling distances. After all patrolling paths are produced, we check whether the differences of these paths are acceptable according to the threshold; if not, the nodes of all paths must be chosen afresh.

4.1.4 Top-view omni-monitoring

In this study, two cameras are fixed on the ceiling and each of them is equipped with a fish-eye lens. That is the reason why the camera is called a *top-view monitoring omni-camera*. At times, there may be some unexpected problems with the vehicles, such as not complying with the user command. The cameras can overlook the patrolling area and so control all actions of the vehicles. If one vehicle is not under control, all vehicles will be stopped. And then the system will send an alarm message to the control center.

4.1.5 Path correction by top-view monitoring omni-camera

Autonomous vehicles used in this study are subject to accumulation of mechanic errors, so they must be localized periodically. About vehicle localizations, house corners, geometric shapes, and object features all may be utilized to localize the vehicle. For the vehicle localization technique proposed in this study, the top-view omni-cameras are utilized. Because the cameras are fixed, images acquired by them are good for analyzing the actual positions of the vehicles. The entire detail of vehicle localization is described in Section 4.2.2.

4.2 Proposed Techniques Used in Patrolling

4.2.1 Optimal randomized patrolling paths for all vehicles

All patrolling paths planned by algorithms proposed in this study have three properties: randomization, optimization, and load balancing. Let the total number of autonomous vehicles and monitoring positions (*MP*s) be n_v and n_m, respectively. In the proposed method for generating random patrolling paths, we divide all *MP*s into n_v groups randomly. Assume that the number of chosen *MP*s for the *i*-th vehicle is n_i, so that the numbers can be represented as (n_1, n_2, ..., n_{n_v}). Each n_i must satisfy two conditions as listed in the following.

Condition 1:

$$n_1 + n_2 + ... + n_{n_v} = n_m - n_v$$

Condition 2:

$$\left\lfloor \frac{n_m - n_v}{n_v} \right\rfloor - Tvalue \le n_i \le \left\lceil \frac{n_m - n_v}{n_v} \right\rceil + Tvalue, i = 1, 2, ..., n_v$$

where *Tvalue* is an adjustable parameter. Because n_v *MP*s are patrolled by n_v vehicles at the end of the *t*-th session, there is no need to visit the n_v *MP*s again in the (t + 1)-th session. That is the reason why "$n_m - n_v$" is included in Condition 1. Additionally, the purpose of Condition 2 is to achieve load balancing among all vehicles. Because we set a threshold parameter *T* to restrict the differences of the patrolling distances, each

n_i dose not have to be equal to the mean $\left\lceil \frac{n_m - n_v}{n_v} \right\rceil$ or $\left\lfloor \frac{n_m - n_v}{n_v} \right\rfloor$. Therefore, we add the parameter *Tvalue* to obtain an upper bound "$\left\lceil \frac{n_m - n_v}{n_v} \right\rceil$+*Tvalue*" and a lower bound "$\left\lfloor \frac{n_m - n_v}{n_v} \right\rfloor - Tvalue$" for all n_i. The parameters *Tvalu*e and *T* are adjustable. If they are smaller, the time taken to determine all patrolling paths is larger, but the loads of all vehicles will be more balanced.

The state of choosing *MP*s for all vehicles can be represented as

$$(C_{n_1}^{n_m - n_v}, C_{n_2}^{n_m - n_v - n_1}, \ldots, C_{n_{nv}}^{n_m - n_v - n_1 - n_2 - \ldots - n_{n_v - 1}}).$$

The combination C_k^n is the number of picking *k MP*s from *n MP*s randomly, defined as

$$C_k^n \equiv \binom{n}{k} = \frac{n!}{k!(n-k)!}. \tag{4.1}$$

For example, assume that the number of *MP*s is "thirteen", the number of vehicles is "three", and the parameter *Tvalue* is "one". By Condition 1, "ten *MP*s" need be divided into "three groups." Besides, the number of each group has an upper bound of "five" and a lower bound of "two" by Condition 2. The three states of the numbers of *MP*s chosen for three vehicles are shown in following.

(1) (5,3,2)

(2) (4,4,2)

(3) (4,3,3)

For state (1), the number of the combination is $C_5^{10} * C_3^5 * C_2^2$=2520. For state (2), the number of the combination is $\frac{C_4^{10} * C_4^6 * C_2^2}{2!}$=1575. For state (3), the number of the combination is $\frac{C_4^{10} * C_3^6 * C_3^3}{2!}$=2100. Hence, the total number of combinations is 2520+1575+2100=6195. It means that the total number of the states of choosing *MP*s for the vehicles randomly is 6195 and the probability of choosing the same state in two continuous sessions is $\frac{1}{6195}$.

As long as one group of *MP*s is determined, we calculate next a path passing all of the *MP*s under the constraint that the distance of the path is the shortest. The *MP*s are the positions where vehicles perform the security monitoring task, and every *MP* is just passed one time in this path. In other words, the requirement is that one vehicle passes each *MP* once (once and only once) and takes the shortest time to accomplish the route. The problem is equal to the *traveling salesman problem* (*TSP*) and the detail to solve it for our application of this study is described in Section 5.2.

To solve the problem by the idea of the TSP, the information of the distance between each pair of *MP*s is needed. In this study, the floor shape of a vehicle patrolling environment is assumed to be composed of rectangular regions. There may be two *MP*s which do not belonging to the same region. If two *MP*s are in different regions, the vehicle might not be able to move along a straight path between them without hitting obstacles. To obtain the distance between every pair of *MP*s, we must judge whether one pair of *MP*s belong to an identical rectangular region. If yes, the distance of this pair is the straight distance between them; else, the straight path between them must be abandoned and a new path with multiple line segments should be planned using some turning points obtained in the learning phase. Because the between-*MP* distance is desired to be the shortest, it can be figured out that the distance may be computed by the Dijkstra's algorithm. The detail for the solution of this problem is described in Section 5.1.

In the proposed system, the reason why each *MP* is just passed one time is that it is wished to patrol these monitored objects *uniformly* in time. That is, the difference between the biggest and the smallest times of *MP*s passed at any moment is desired to be *one or zero*. In this sense, each *MP* will have been visited *t* times at the end of the *t*-th session of security patrolling. And then the system will calculate new patrolling paths for the next session. A flowchart is illustrated in Figure 4.1, and the detail of obtaining the patrolling paths is described as an algorithm in the following.

Algorithm 4.1 *Calculating all patrolling paths.*

Input: *MP*s, the number n_m of *MP*s, the number n_v of vehicles, the points *P*s where the *MP*s are patrolled by all vehicles at the end of the front session, the distance between every pair of *MP*s, threshold parameters *T* and *Tvalue*.

Output: Optimal patrolling paths for the vehicles.

Steps:

Step 1. Divide the number "$n_m - n_v$" into n_v groups according to the following conditions randomly and list all states, such as $(n_1, n_2, ..., n_{n_v})$.

Condition 1:

$n_1 + n_2 + ... + n_{n_v}$

Condition 2:

$$\left\lfloor \frac{n_m - n_v}{n_v} \right\rfloor - Tvalue \le n_i \le \left\lceil \frac{n_m - n_v}{n_v} \right\rceil + Tvalue,\ i = 1, 2, ..., n_v$$

Step 2. Choose *MP*s, not including *P*s, randomly according to the numbers $(n_1, n_2, ..., n_{n_v})$.

Step 3. Calculate every patrolling path passing all chosen *MP*s such that the distance of the path is the shortest.

Step 4. Check whether all distance differences between each pair of paths conform to the threshold parameter T. If yes, the paths are determined. Else, repeat Step 1.

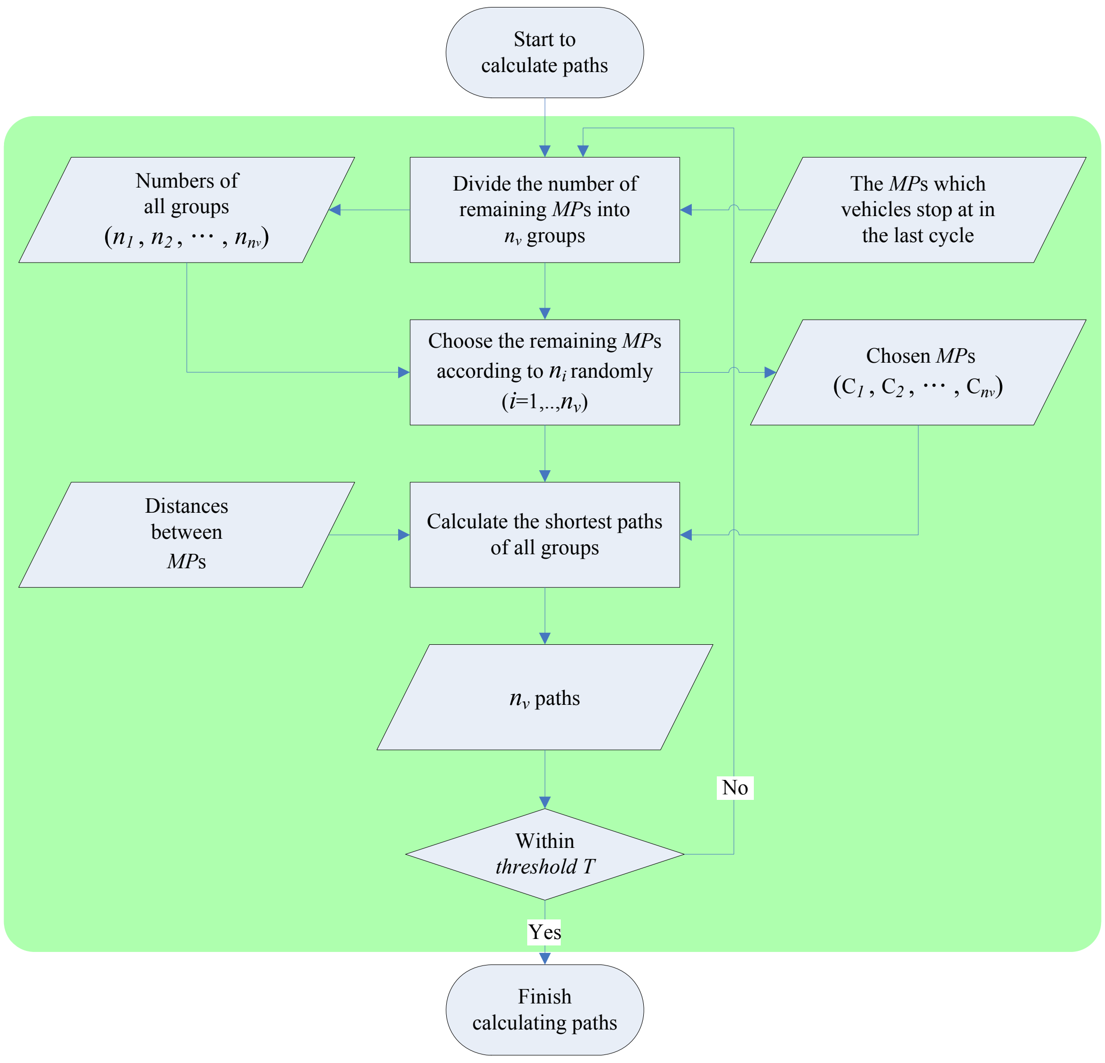

Start to calculate paths
Numbers of all groups ($n_1, n_2, \cdots, n_{nv}$)
Divide the number of remaining *MP*s into n_v groups
The *MP*s which vehicles stop at in the last cycle
Choose the remaining *MP*s according to n_i randomly (i=1,..,n_v)
Chosen *MP*s ($C_1, C_2, \cdots, C_{nv}$)
Distances between *MP*s
Calculate the shortest paths of all groups
n_v paths
No
Within threshold *T*
Yes
Finish calculating paths

Figure 4.1 A flowchart of determining all paths for vehicles.

4.2.2 Guidance of vehicles by localization and monitoring using top-view omni-cameras

Before introducing the proposed scheme for guidance of vehicles, we take an illustration of all coordinate systems used in the system. They are the image coordinate system, the global coordinate system, and the vehicle coordinate system. By these coordinate systems, it is easily to know the position of a vehicle. The definitions of the three coordinate systems are described in the following and a graphic illustration is shown in Figure 4.2.

(1) The image coordinate system (ICS, denoted as *u*-*v*):

The coordinate system is used for the image acquired by the top-view omni-cameras fixed on the ceiling, in which the *u*-*v* plane is parallel to the floor where vehicles navigate. If the image is displayed in the user interface, the positive direction of the *u*-axis is from left to right and the positive direction of the *v*-axis is from top to bottom. The origin O_{uv} is the upper left corner of the image.

(2) The global coordinate system (GCS, denoted as *X*-*Y*):

This coordinate system is in the 3-D global space where the vehicles navigate. Because the top-view omni-cameras are fixed on the ceiling, the distance between each camera and the floor of the global space is fixed. Besides, we only need to know where the vehicles are located at nay moment, so there are merely two axes in the coordinate system. To simplify related computations, the positive directions of the *X*- and *Y*-axes are the same as the *u*- and *v*-axes in the CCS, respectively. The origin O_{XY} is the upper left corner of the patrolling environment.

(3) The vehicle coordinate system (VCS, denoted as *x*-*y*):

In this system, the *x*-*y* plane is also parallel to the floor. The positive directions of the *x*- and *y*- axes are the front direction and the leftward direction of the vehicle, respective. The origin of the VCS is where the vehicle starts its navigation.

For vehicle localization, the *position* and the *direction angle* of a vehicle must be calibrated after the vehicle moves a fixed distance because the vehicle is subject to accumulation of mechanic errors. In the proposed system, a top-view omni-camera is utilized to acquire the current location of the vehicle. To reduce the cycle time of the navigation session, we only calculate a region, whose center is the odometer value of the vehicle and whose width is a parameter W, to find out the centroid of the vehicle in the image. By the way, the negative influence of noise is also decreased. As an example, the black region with a red rectangle frame shown in Figure 4.3 is the range of calculation.

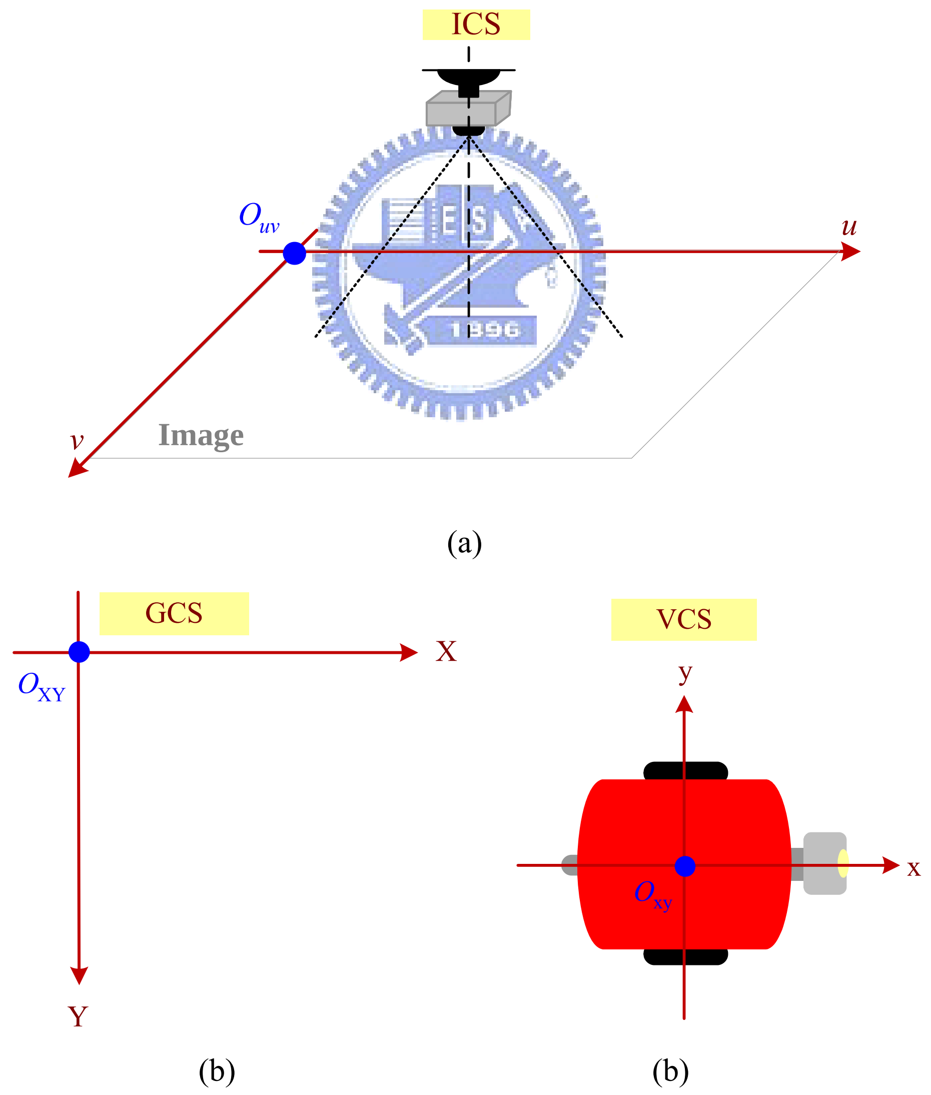


Figure 4.2 Coordinate systems used in this study. (a) ICS (b)GCS (c)VCS

Before all vehicles perform the patrolling task, the top-view omni-cameras capture an image of the floor without vehicles and unnecessary objects as a *background*. An example of background images is shown in Figure 4.4. Because the number of the cameras used in the study is more than one, when a vehicle needs to be located, the system must judge which camera will do the job according to the odometer value of the vehicle. We translate the coordinate of the vehicle from the VCS into the GCS and then judge the camera view which includes the vehicle.

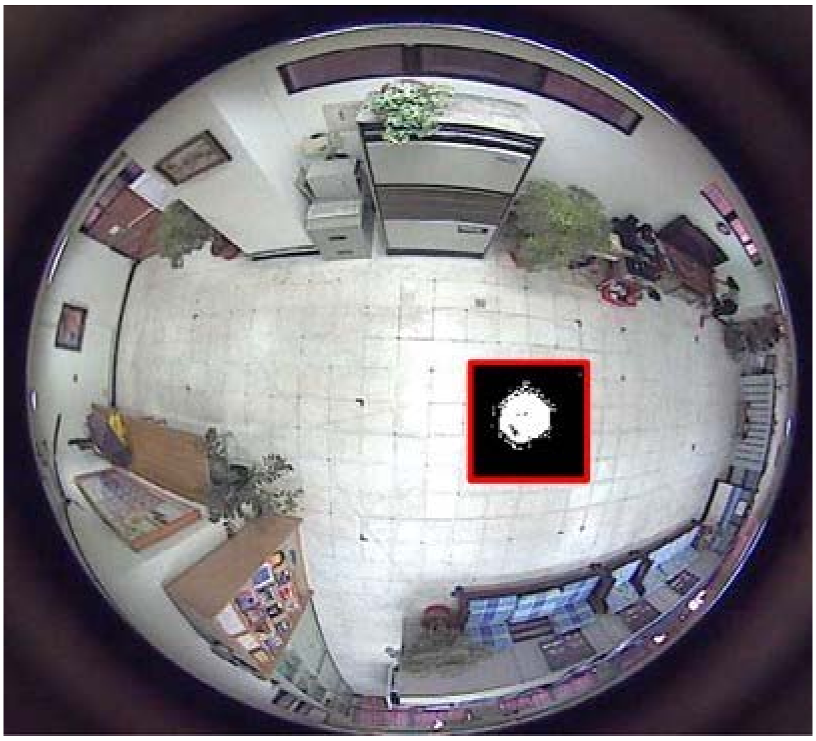


Figure 4.3 Vehicle localization from a top-view omni-camera.

To acquire the position of the vehicle, the camera must capture the current image as a *foreground*. An example of foreground image is shown in Figure 4.5.

By subtracting the foreground from the background, we can obtain all differences in the two images. Because there is a lot of noise in the patrolling environment, such as light variations, we set an appropriate threshold parameter *T_Diff* to threshold the difference image to eliminate noise. If the difference value of a pixel is larger than the parameter *T_Diff*, the pixel will be recorded as "1"; else it will be recorded as “0”. At the end, we will obtain a binary image *B_IM*. Furthermore, we apply consecutively two methods of morphology to decrease the interference of noise. One is *erosion* and the other is *dilation*. The equations of them are described in the following where *A* and *B* are sets in Z^2 and all elements of them are zero (false) or one (true); $\hat{B}$ is the complement of the set *B* and its origin is *z*:

$$A \oplus B = \{z \mid (\hat{B})_z \cap A \neq \varnothing\}; \tag{4.2}$$

$$A \Theta B = \{z \mid (B)_z \subseteq A\}. \tag{4.3}$$

We denote the *dilation* of *A* by *B* by $A \oplus B$ for which if there is at least one element overlap between $\hat{B}$ and *A*, then *z* is set true. The result of dilation is using *B* as a mask translated by *z* over the set *A*. The *erosion* of *A* by *B* is denoted $A \Theta B$ for which if *B* is contained in *A*, then *z* is set true. The result of erosion is also using *B* as a mask translated by *z* over the set *A*.

As we obtain a binary image *B_IM*, the noise which is smaller or bigger than the vehicle too much can be eliminated by the method of *erosion* using some squares as the mask. For example, given a binary image *A* composed of the square *D* of size 1 pixel on the side and a mask *B* also being a square of size 3 pixels on the side, erosion of *A* by *B* results in the square *D* being eliminated.

After erosion, we can perform the method of *dilation* to repair some holes in the range of the shape of the vehicle. By the way, if the shape of the vehicle can be more complete, then the obtained position will also be more precise.

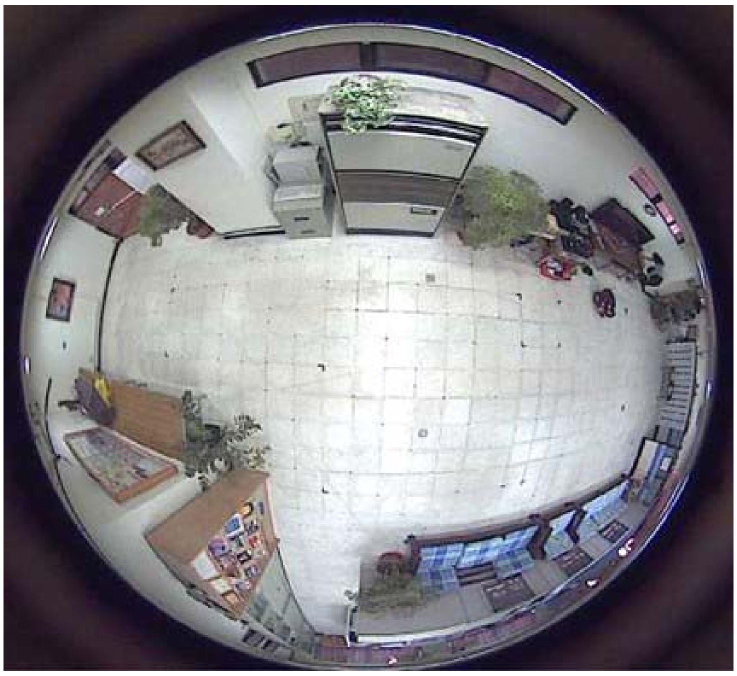


Figure 4.4 A background image taken by a top-view omni-camera.

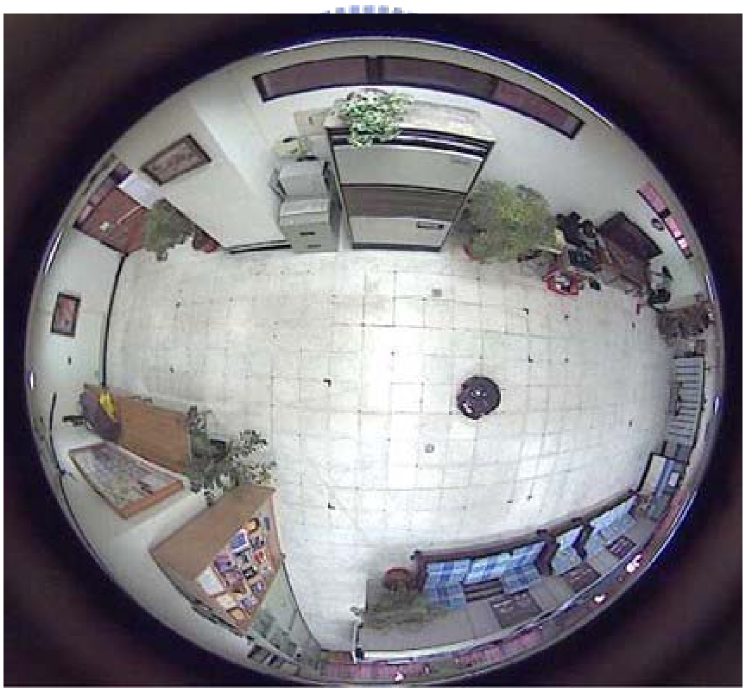


Figure 4.5 A foreground taken by a top-view omni-camera.

As the next step, we use the method of *connected component labeling*. If the number of components is larger than one, it is necessary to find out the one component which is much like the shape of the vehicle by the number of the pixels in the component. Furthermore, if no component is found, the state is that there is one vehicle which is not under control, for which the system will send a message to the control center. When only one component is detected, we calculate the centroid of the component, which is exactly the position of the vehicle in the CCS. The position then is transformed into one in the GCS by the point-correspondence technique integrated with an image interpolation method described in Section 3.2.2. For Figure 4.3, the result of finding the position of a vehicle is shown in Figure 4.6. The entire detail is described as Algorithm 4.2.

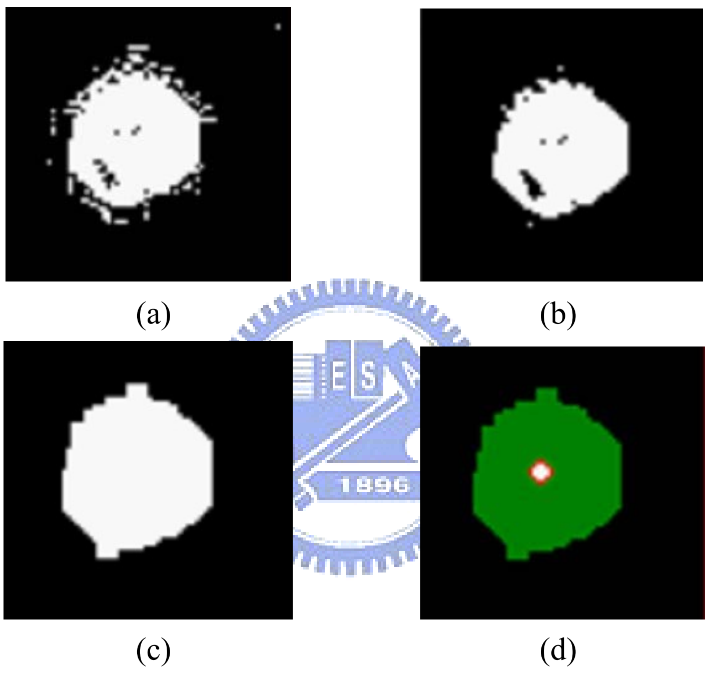


Figure 4.6 Finding the position of a vehicle. (a) A binary image which is part of the image shown in Figure 4.3. (b) Erosion of (a) with a square mask. (c) Dilation of (b) with a square mask. (d) The connected component of (c) and the computed centroid (the white circle).

Algorithm 4.2 *Calculating the position, in the GCS, of a vehicle by top-view omni-cameras.*

Input: A background image *Back_IM*, the odometer value of the vehicle *Va*, the width *W* of the calculation range, and a threshold parameter *T_Diff*

Output: The position, in the GCS, of the vehicle.

Steps:

Step 1. Judge which camera will do the job to calculate the centroid of the vehicle by *Va*.

Step 2. Capture a foreground image *Foer_IM* by the camera determined from Step 1.

Step 3. Translate *Back_IM* and *Foer_IM* into gray images *Back_GaIM* and *Foer_GaIM*.

Step 4. Subtract *Foer_GaIM* from *Back_GaIM*, in a region whose center is *Va* and whose width is *W*, to obtain another image *D*.

Step 5. Acquire a binary image *Bi_IM*.

In the binary image *Bi_IM*, “1” means that this pixel in the image *D* is bigger than *T_Diff*.

Step 6. Perform *erosion* of *Bi_IM* with a square mask and to obtain a new image *Ero_IM*.

Step 7. Perform *dilation* of *Ero_IM* with a square mask to obtain a new image *Dila_IM*.

Step 8. Find out one connected component *Co_comp* in *Dila_IM*.

Step 9. Calculate the centroid *c* of the component *Co_comp*.

Step 10. Translate *c* into one in the GCS.

Because vehicle navigation must utilize the direction angle of the vehicle, it is important to ensure the accuracy of the direction angle. The reason why the direction angle must be corrected is illustrated in Figure 4.7. A vehicle starts at point *O* and moves forward a distance. The position *B* is desired, but the vehicle arrives at position *A*.

At the moment, the direction angle of the vehicle is recorded as zero. In reality, the correct value is θ_1. Because the vehicle suffers from mechanic errors, a user’s command to move the vehicle straightly for a distance will result in a curved trajectory. In this study, we utilize two continuous correct positions to acquire a direction angle, such as θ_2 in Figure 4.7, and the detail is described in Section 4.3. Because the distance between two continuous localizations is short, the curve path is close to a straight line. We can correct the direction angle of the vehicle by θ_2. Finally, the detail of localizing and monitoring vehicles is described as Algorithm 4.3 and the flowchart of the entire process is shown in Figure 4.8.

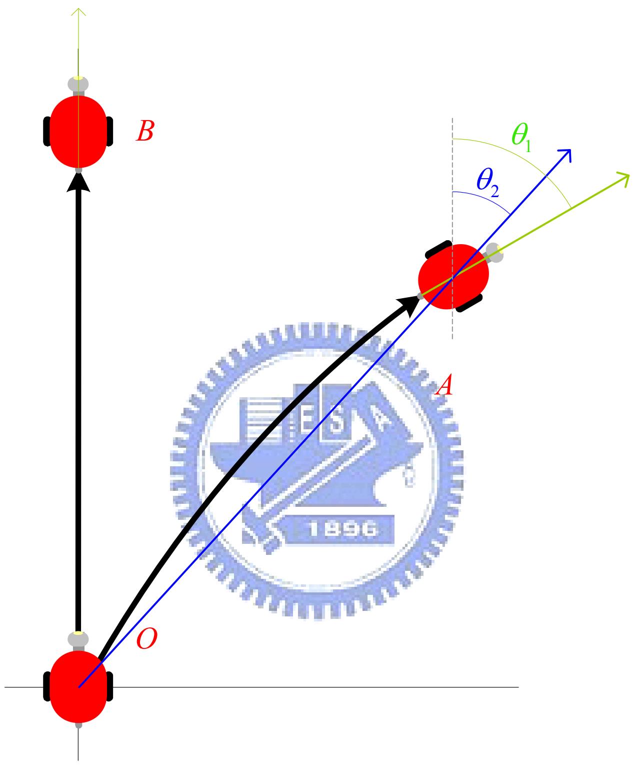


Figure 4.7 Illustration of direction angles of the vehicle.

Algorithm 4.3 *Vehicle localization and monitoring.*

Input: A background image *IM_ Back*, the odometer value of a vehicle *Va*, and a threshold parameter *T_Loss*.

Output: Location of the vehicle.

Steps:

Step 1. Determine which camera needs to do the job of vehicle localization by *V*a.

Step 2. Capture the foreground image *IM_Fore* by the camera.

Step 3. Calculate the centroid *c* of the vehicle in one image using *IM_ Back* and *IM_Fore*.

Step 4. Check whether the point *c* is found.

If yes, go to Step 6.

Step 5. Check whether the number of cycle times not finding the vehicle is more than *T_Loss*.

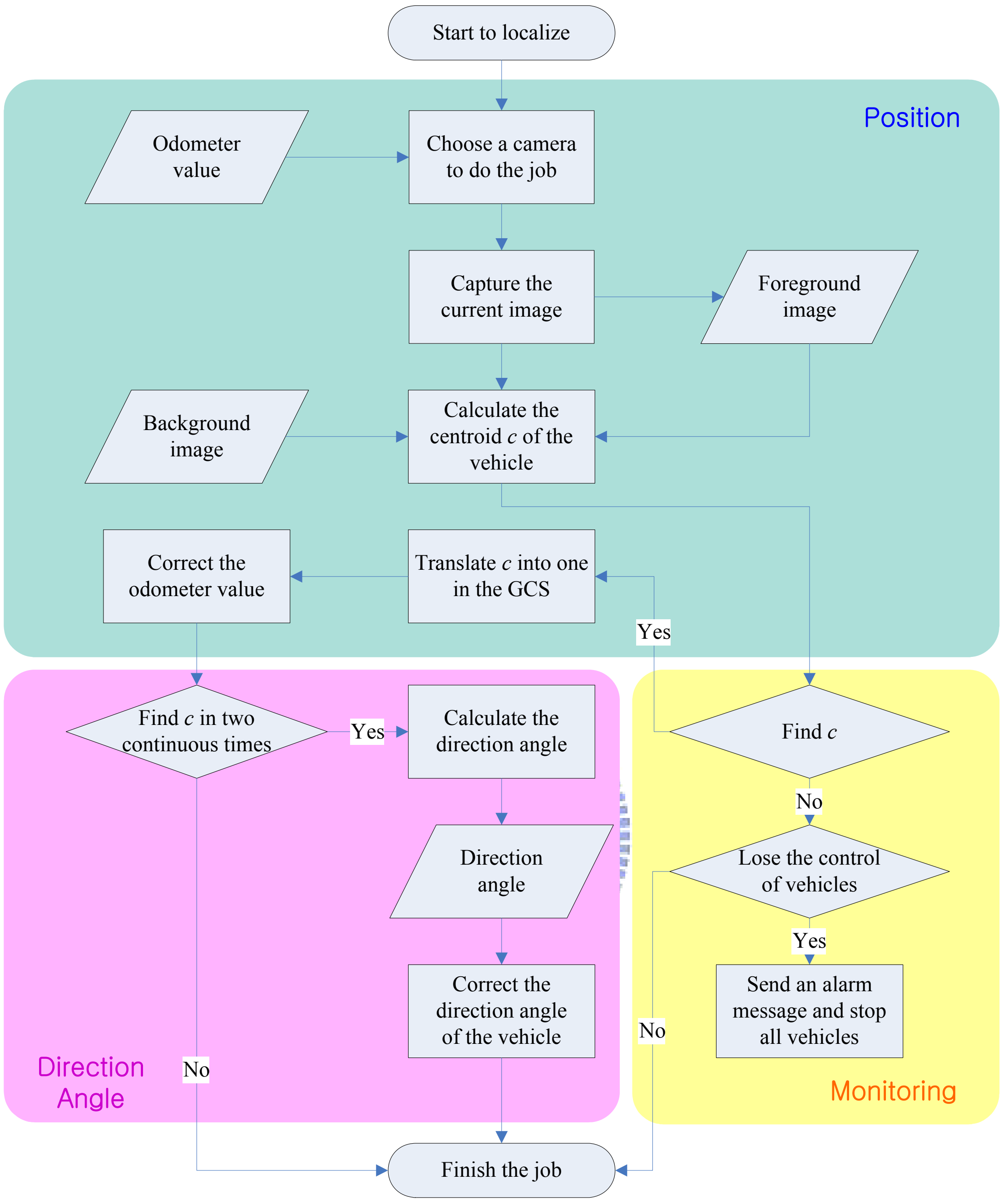
If yes, the system sends an alarm message to the security center and stops all vehicles.

Else, move the vehicle toward the goal.

Step 6. Translate the point *c* into one in the GCS and correct the odometer value of the vehicle by the point.

Step 7. Check whether the correct position of the vehicle is found in two continuous cycles of navigation.

If yes, the two continuous correct positions are utilized to calculate the direction angle, and take the angle value to replace the original direction angle of the vehicle.



Start to localize
Position
Odometer value
Choose a camera to do the job
Capture the current image
Foreground image
Background image
Calculate the centroid *c* of the vehicle
Correct the odometer value
Translate *c* into one in the GCS
Yes
Find *c* in two continuous times
Yes
Calculate the direction angle
Find *c*
No
Direction angle
Lose the control of vehicles
Yes
Correct the direction angle of the vehicle
Send an alarm message and stop all vehicles
No
Direction Angle
No
Monitoring
Finish the job

Figure 4.8 Flowchart of localizing and monitoring vehicles.

4.2.3 Avoiding collisions between vehicles

Because the vehicles are located constantly by the top-view omni-cameras, the odometer values can be utilized to avoid collisions among vehicles. The system sets a timer to detect whether two vehicles are too close. If yes, their patrolling paths are changed by calculating some *passing points* and inserting them into the original path. By these passing points, the distance between the two vehicles can be drawn apart. The detail about the proposed collision avoiding technique is described in Chapter 6.

4.3 Detailed Process for Security Patrolling by Vehicle Navigation

In the navigation phase, the vehicles navigate along assigned patrolling paths by arriving at each node orderly. The types of nodes on a path include *monitoring point*, *turning point*, and *passing point*. Assume that N is the set of nodes on a patrolling path and that an element n_i in N means the i-th node passed by a vehicle. If a point $m = (m_x, m_y)$ is the current position of a vehicle, we can utilize the goal node $n_i = (x_i, y_i)$, where the vehicle wants to arrive at, to acquire a direction vector $\overrightarrow{W_i}$. The equation is shown in Eq. (4.3). The direction angle θ_V of a vehicle is an included angle between the current direction vector of the vehicle and the positive direction of the x-axis in the VCS, as shown in Figure 4.9.

We calculate an acute angle θ by the *cosine formula* and utilize the angle to obtain θ_V in the following way:

(1) If the direction vector $\overrightarrow{W_i}$ belongs to the first quadrant, $\theta_V = \theta$.

(2) If the direction vector $\overrightarrow{W_i}$ belongs to the second quadrant, $\theta_V = 180^\circ - \theta$.

(3) If the direction vector $\overrightarrow{W_i}$ belongs to the third quadrant, $\theta_V = -180^\circ + \theta$.

(4) If the direction vector $\overrightarrow{W_i}$ belongs to the fourth quadrant, $\theta_V = -\theta$.

The relation between θ and θ_V is shown in Figure 4.10.

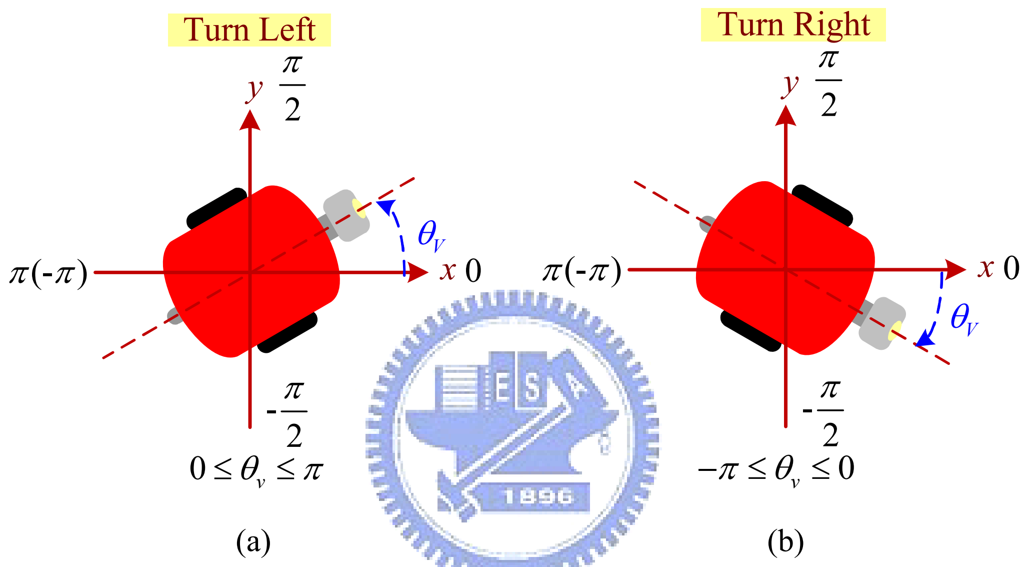


Figure 4.9 A direction angle θ_V. (a) Left ($0 \le \theta_v \le \pi$). (b)Right ($-\pi \le \theta_v \le 0$).

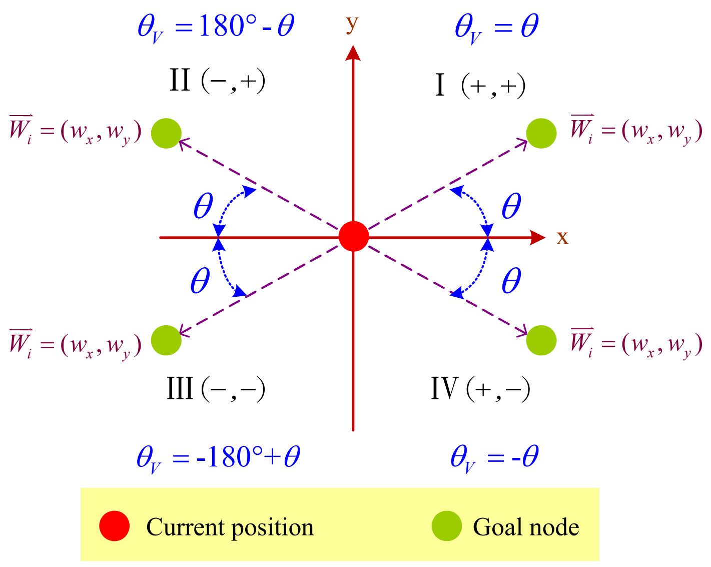


Figure 4.10 The relation between θ and θ_V.

The acute angle θ is an included angle between the direction vector $\overrightarrow{W_i}$ and the x-axis in the VCS, and the equation is shown in Eq. (4.4). If one element of $\overrightarrow{W_i}$ is negative, it will be transformed into positive one. As the direction angle θ_V is obtained, the difference between θ_V and the original direction angle θ_{odo} is exactly the rotation angle θ_{turn}.

The proposed system utilizes nodes to guide the vehicles. By the current node and the goal node, we can obtain the direction vector $\overrightarrow{W_i}$ and then the rotation angle θ_{turn}. The vehicles turn to the angle θ_{turn} and move forward. Such actions enable vehicles to arrive at the goal node. In the period of navigation, the vehicles must be located constantly. Therefore, we set a distance parameter d. When one vehicle has moved the distance d, it must be located. Furthermore, if the vehicle arrives at a goal node which is a monitoring point, the direction angle of the vehicle must be adjusted as one θ_{moni} obtained in the learning phase and then performs the security monitoring task. The algorithm is shown in the following and the flowchart is shown in Figure 4.11.

Algorithm 4.4 *Navigation and monitoring tasks.*

Input: The current position $m = (m_x, m_y)$, the goal node $n_i = (x_i, y_i)$, and distance parameter d.

Output: The vehicle moves the distance d toward the goal node.

Steps:

Step 1. Locate the vehicle including correcting the position and the direction angle.

Step 2. Calculate the direction vector $\overrightarrow{W_i}$ from the node m to the node n_i by the following equation:

$$\overrightarrow{W_i} = \begin{bmatrix} w_x \\ w_y \end{bmatrix} = \begin{bmatrix} x_i - m_x \\ y_i - m_y \end{bmatrix}. \tag{4.3}$$

Step 3. Transform $\overrightarrow{W_i}$ into $\overrightarrow{W_i}'$ each element of which is the absolute value with respect to the element in $\overrightarrow{W_i}$.

Step 4. Calculate the acute angle θ by the following equation:

$$\overrightarrow{W_i}' = abs(\overrightarrow{W_i});$$
$$\theta = \cos^{-1}\left(\frac{W_i' \bullet (1,0)}{|W_i'||(1,0)|}\right). \tag{4.4}$$

Step 5. Calculate the direction angle θ_V by the following rules.

(1) If the direction vector $\overrightarrow{W_i}$ belongs to the first quadrant, $\theta_V = \theta$.

(2) If the direction vector $\overrightarrow{W_i}$ belongs to the second quadrant, $\theta_V = 180^\circ - \theta$.

(3) If the direction vector $\overrightarrow{W_i}$ belongs to the third quadrant, $\theta_V = -180^\circ + \theta$.

(4) If the direction vector $\overrightarrow{W_i}$ belongs to the fourth quadrant, $\theta_V = -\theta$.

Step 6. Calculate the rotation angle as $\theta_{\text{turn}} = \theta_V - \theta_{odo}$.

Step 7. Turn the vehicle leftward for the angle θ_{turn}.

Step 8. Check whether the distance d is bigger than the distance between the current node and the goal node.

If yes, the vehicle moves the distance d forward.

Else, the vehicle moves to the goal node.

Step 9. Check whether the vehicle arrives at a monitoring point.

If no, perform this Algorithm 4.4 again.

Step 10. Read the direction angle θ_{moni} at the monitoring point.

Step 11. Calculate the rotation angle as $\theta_{\text{turn}} = \theta_{\text{moni}} - \theta_{odo}$.

Step 12. Turn the vehicle leftward for the angle θ_{turn}.

Step 13. Perform the security monitoring task.

Step 14. Read the next goal node and perform this Algorithm 4.4 again.

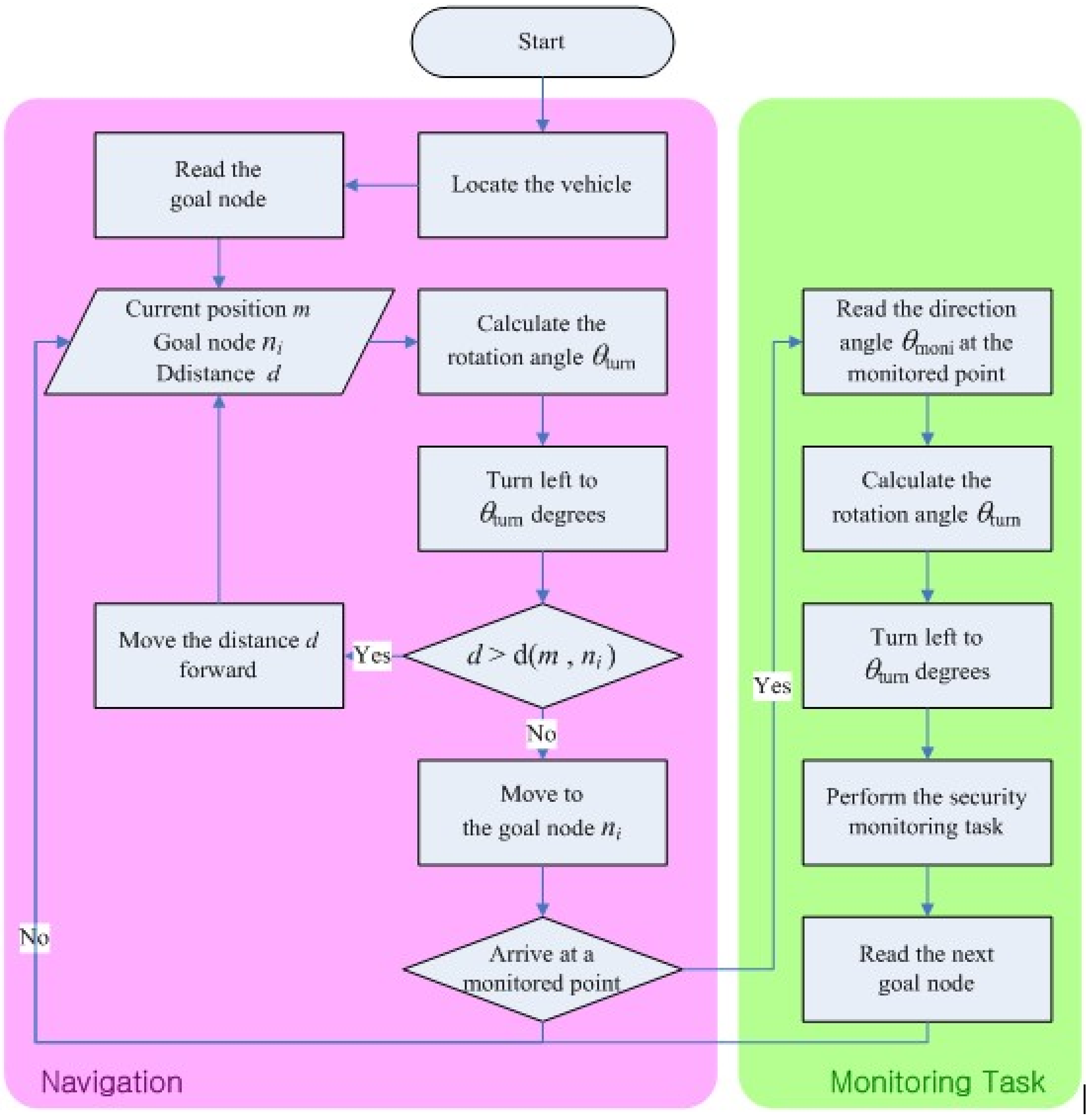

Start
Locate the vehicle
Read the goal node
Current position m
Goal node n_i
Ddistance d
Calculate the rotation angle θ_{turn}
Turn left to θ_{turn} degrees
$d > d(m, n_i)$
Yes
Move the distance d forward
No
Move to the goal node n_i
Arrive at a monitored point
Yes
No
Read the direction angle θ_{moni} at the monitored point
Calculate the rotation angle θ_{turn}
Turn left to θ_{turn} degrees
Perform the security monitoring task
Read the next goal node
Navigation
Monitoring Task

Figure 4.11 Flowchart of Navigation and monitoring tasks.

Chapter 5
Planning of Optimal Randomized Patrolling Paths for Vehicles

5.1 Introduction

In the proposed system, patrolling paths are designed to be optimal, random, and load-balanced for all autonomous vehicles in the senses mentioned previously. All monitoring points on these paths are chosen randomly. Because some monitoring points might belong to different rectangular regions, the turning points are utilized to enable the vehicles to move between any pair of monitoring points without collisions with walls. To optimize the patrolling paths, all distances between monitoring points must be the shortest. Therefore, we calculate the distances by *Dijkstra's algorithm*. The detail is described in Section 5.2.

To patrol all monitored objects uniformly, each monitoring point appears only once in one patrolling session. With all distances between pairs of monitoring points, we utilize the idea of the *traveling salesman problem* to obtain the optimal patrolling paths, as mentioned previously. In such a way, all autonomous vehicles can take shorter time to accomplish the security patrolling task in each session. The detail is described in Section 5.3.

5.2 Calculation of Paths between Monitoring Points by Dijkstra's Algorithm

5.2.1 Review of Dijkstra's algorithm

Dijkstra's algorithm can be adopted to solve a single-source shortest-paths problem on a weighted, directed graph G = (V, E), in which V is the set of vertices and E is the set of edges. The algorithm is only feasible for the case that all edge weights are nonnegative. By the algorithm, the shortest paths and weights from a source s to other vertices can be obtained. In the algorithm, there are three symbol definitions:

(1) $w(u, v)$ represents the *weight* of the edge (u, v);

(2) d[v] represents the *shortest-path estimate* from the source s to the vertex v; and

(3) π[v] represents the *predecessor* of v in the shortest path from the source s to the vertex v.

The main concept is that the all sub-paths of the shortest path are also the shortest paths. Therefore, it uses the technique of *relaxation* [24] by a triangle inequality. The process is to check constantly whether the shortest path to v can be improved by going through u. If yes, then update d[v] and π[v]. Dijkstra's algorithm [24] is shown below.

Algorithm 5.1 *Dijkstra's algorithm*.

Input: A directed graph G including vertices V and edges E, all weights with respect to edges E, a source s, and an empty set S.

Output: The shortest paths and weights from s to other vertices.

```
Steps:
Step 1.   Set all d[$v$] = $\infty$ and all $\pi$[$v$] = nil.
Step 2.   Set d[$s$] = 0.
Step 3.   Find out vertex $u$ whose shortest-path estimate d[$u$] is the smallest.
Step 4.   Put the vertex $u$ into the set $S$.
Step 5.   Find out all vertices $v$ which are adjacent to the vertex $u$ from $E$ and
          which are not in the set $S$.
Step 6.   Check whether d[$v$] > d[$u$] + $w(u, v)$.
          If yes, update d[$v$] = d[$u$] + $w(u, v)$ and $\pi$[$v$] = $u$.
Step 7.   Repeat Step 3 if the set $S$ do not contain all vertices $V$.
```

5.2.2 Proposed technique for generation of partial patrolling paths

To calculate the distance between every pair of monitoring points, we adopt *Dijkstra's algorithm*. At first, we check whether two monitoring points belong to an identical rectangular region. If yes, the distance between them is set equal to the straight line distance; else, we calculate the shortest path by passing through some turning points without colliding with walls.

To meet the assumptions in Dijkstra's algorithm, the patrolling environment is taken to be a directed graph G. All turning points and monitoring points are regarded as the set V of vertices in the graph G. If there is a rectangular region which two turning points belong to or which both of a turning point and a monitoring point belong to, then there exists an edge between the two and it is taken to be one element of the set E of edges in the graph G. The directions of all edges are two-way. The straight line distances of all edges in the set E are the set W of weights.

To calculate the shortest path between two monitoring points which belong to different regions, we set one of the two points as a *source point* and the other as an *end point*, at first. By Dijkstra's algorithm, the shortest path from the source point to the end point can be obtained and its weight is exactly the distance between them. The algorithm of processing all pairs of monitoring points is described in following.

Algorithm 5.2 *Computing distances between two monitoring points*.

Input: Rectangular regions R, monitoring points, and turning points.

Output: All the shortest distances and paths between all pairs of monitoring points.

Steps:

Step 1. Produce the set V which is composed of all monitoring points and turning points.

Step 2. Produce the set E, in which every element means that it connects two points which belong to an identical rectangular region.

Step 3. Produce the set W, in which each element is the straightly line distance with respect to the edge in the set E.

Step 4. Check whether two monitoring points M_i and M_j belong to an identical region.

If yes, the distance between them is exactly the straightly line distance and repeat Step 4 until the distances between all pairs of monitoring points are obtained.

Else, perform *Dijkstra's algorithm* with the source point M_i and the end point M_j.

Step 5. Record the weight and the order of the turning points passed by from the source point M_i and the end point M_j, according to the result from *Dijkstra's algorithm*.

Step 6. Repeat Step 4 until the distances between all pairs of monitoring points are obtained.

As an example, a patrolling environment is shown in Figure 5.1, in which there are five rectangular regions, five turning points N_1 through N_5, and two monitoring points M_1 and M_2.

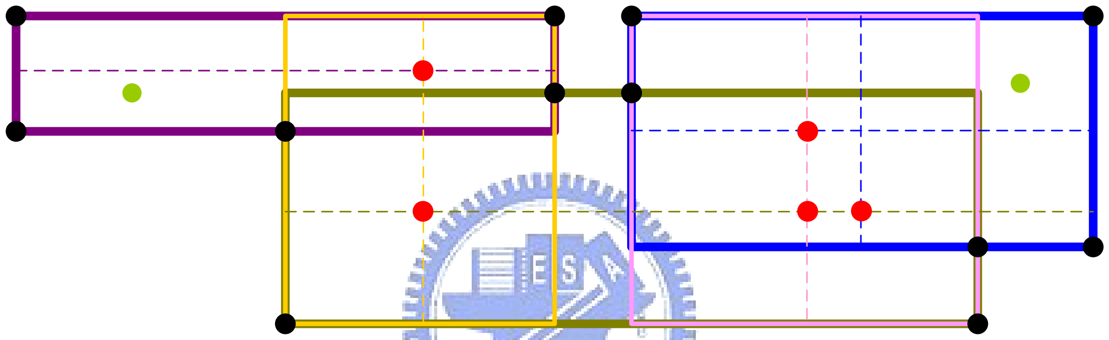


Figure 5.1 A patrolling environment.

Because M_1 and M_2 belong to different rectangular regions, the distance between them is calculated by *Dijkstra's algorithm*. Before performing the algorithm, we transform the patrolling environment into a graph $G = (V, E)$, as shown in Figure 5.2. The set V contains N_1 through N_5 and M_1 through M_2; the set E includes all the black lines which are two-way. M_1 is the source point and M_2 is the end point.

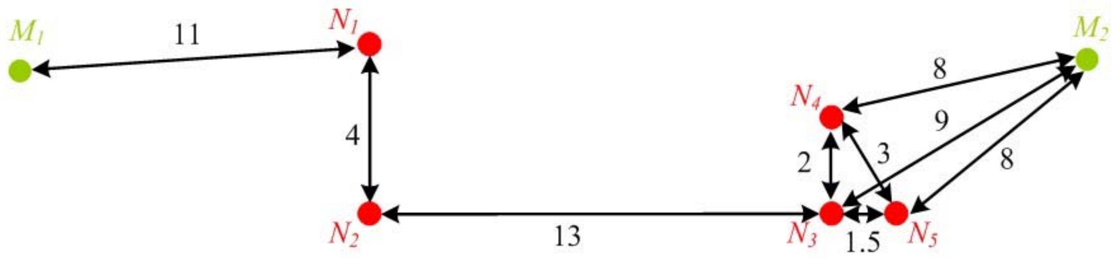
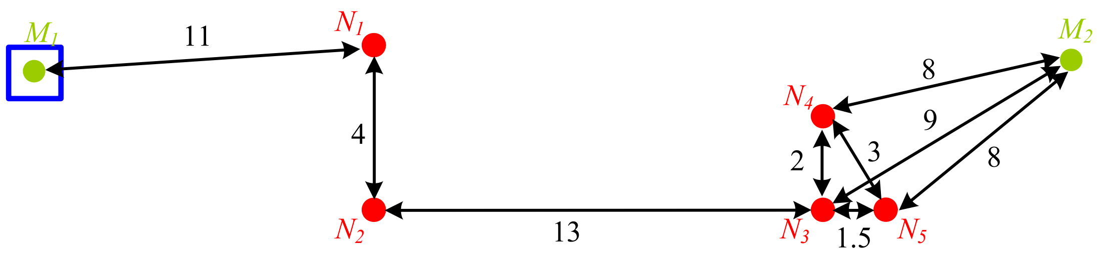


Figure 5.2 The graph G from Figure 5.1.

The results of each process are shown in Figure 5.3 through Figure 5.9 step by

N_2

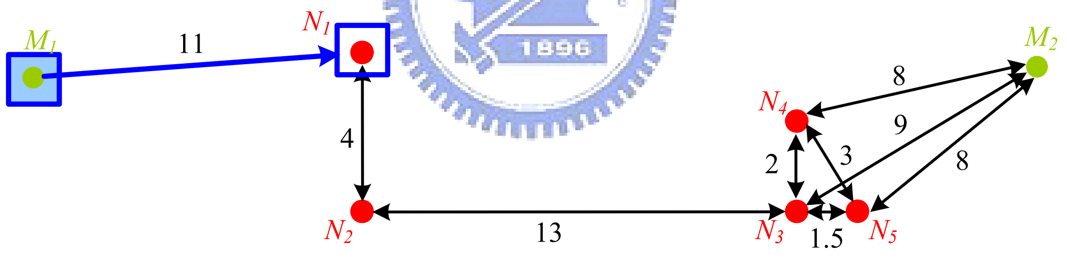
step. Because M_1 is the source point, it has the smallest distance from the source point in the first cycle.



v	M_1	M_2	N_1	N_2	N_3	N_4	N_5
d[v]	$\boxed{0}$	∞	∞	∞	∞	∞	∞
π[v]		nil	nil	nil	nil	nil	nil

Figure 5.3 The first cycle.

In the second cycle, N_1 has the smallest distance within non-chosen vertices. Therefore, the shortest path for the source point is $M_1 \rightarrow N_1$ and the distance is 11.

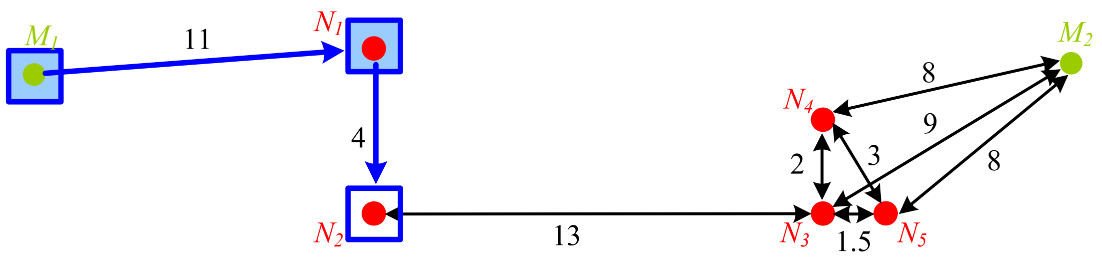


v	M_1	M_2	N_1	N_2	N_3	N_4	N_5
d[v]	$\boxed{0}$	∞	$\boxed{11}$	∞	∞	∞	∞
π[v]		nil	M_1	nil	nil	nil	nil

Figure 5.4 The second cycle.

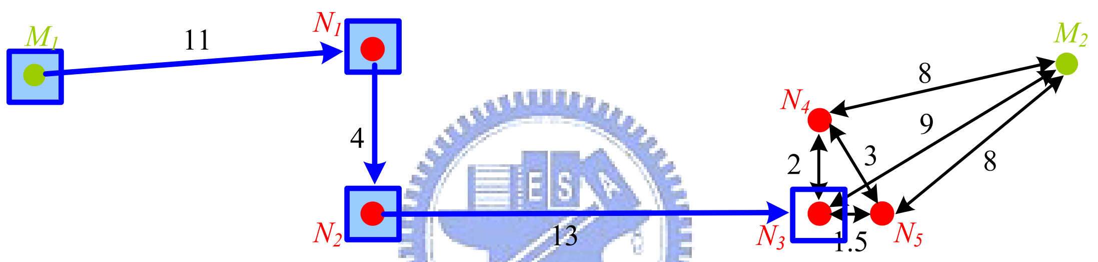
In the third cycle, N_2 has the smallest distance within the non-chosen vertices. Therefore, the shortest path for the source point is $M_1 \rightarrow N_1 \rightarrow N_2$ and the distance is 15.

In the fourth cycle, N_3 has the smallest distance within non-chosen vertices. Therefore, the shortest path for the source point is $M_1 \rightarrow N_1 \rightarrow N_2 \rightarrow N_3$ and the distance is 28.



v	M_1	M_2	N_1	N_2	N_3	N_4	N_5
d[v]	$\boxed{0}$	∞	$\boxed{11}$	$\boxed{15}$	∞	∞	∞
π[v]		nil	M_1	N_1	nil	nil	nil

Figure 5.5 The third cycle.

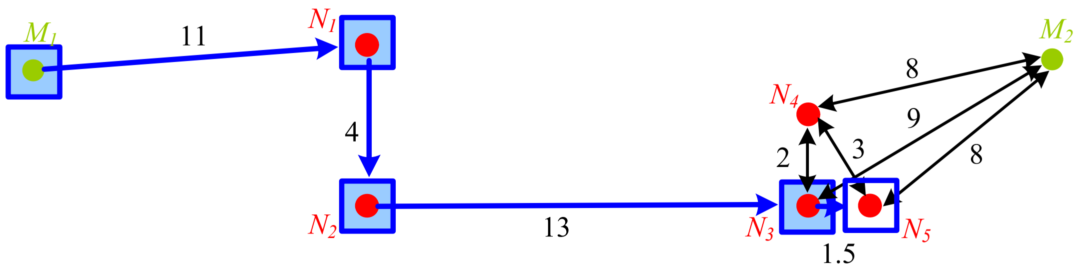


v	M_1	M_2	N_1	N_2	N_3	N_4	N_5
d[v]	$\boxed{0}$	∞	$\boxed{11}$	$\boxed{15}$	$\boxed{28}$	∞	∞
π[v]		nil	M_1	N_1	N_2	nil	nil

Figure 5.6 The fourth cycle.

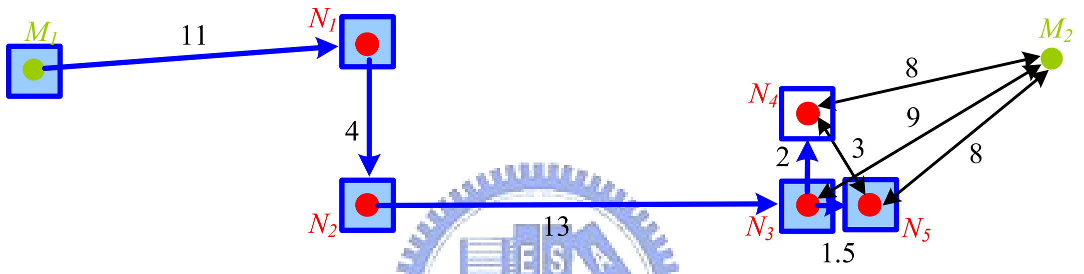
In the fifth cycle, N_5 has the smallest distance within the non-chosen vertices. Therefore, the shortest path for the source point is $M_1 \rightarrow N_1 \rightarrow N_2 \rightarrow N_3 \rightarrow N_5$ and the distance is 29.5.

In the sixth cycle, N_4 has the smallest distance within the non-chosen vertices. Therefore, the shortest path for the source point is $M_1 \rightarrow N_1 \rightarrow N_2 \rightarrow N_3 \rightarrow N_4$ and the distance is 30.



v	M_1	M_2	N_1	N_2	N_3	N_4	N_5
d[v]	$\boxed{0}$	37	$\boxed{11}$	$\boxed{15}$	$\boxed{28}$	30	$\boxed{29.5}$
π[v]		N_3	M_1	N_1	N_2	N_3	N_3

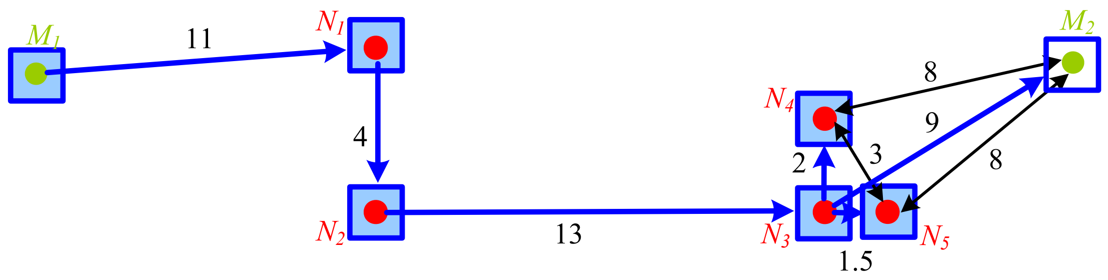
Figure 5.7 The fifth cycle.



v	M_1	M_2	N_1	N_2	N_3	N_4	N_5
d[v]	$\boxed{0}$	37	$\boxed{11}$	$\boxed{15}$	$\boxed{28}$	$\boxed{30}$	$\boxed{29.5}$
π[v]		N_3	M_1	N_1	N_2	N_3	N_3

Figure 5.8 The sixth cycle.

In the seventh cycle, M_2 has the smallest distance within the non-chosen vertices. Therefore, the shortest path for the source point is $M_1 \rightarrow N_1 \rightarrow N_2 \rightarrow N_3 \rightarrow M_2$ and the distance is 37. Finally, the distance between the pair of monitoring points M_1 and M_2 is exactly 37. Furthermore, the path from M_1 to M_2 is $M_1 \rightarrow N_1 \rightarrow N_2 \rightarrow N_3 \rightarrow M_2$ and the path from M_2 to M_1 is $M_2 \rightarrow N_3 \rightarrow N_2 \rightarrow N_1 \rightarrow M_1$. All results of the above derivations must be recorded.



v	M_1	M_2	N_1	N_2	N_3	N_4	N_5
d[v]	0	37	11	15	28	30	29.5
π[v]		N_3	M_1	N_1	N_2	N_3	N_3

Figure 5.9 The seventh cycle.

5.3 Calculation of Optimal Randomized Patrolling Paths by Finding Hamiltonian Paths

5.3.1 Review of Traveling-Salesman Problem (TSP)

The definition of the traveling-salesman problem is that a salesman must visit *n* cities and wishes to visit each city exactly once with the minimum cost. Furthermore, he finishes at the city where he starts from. The problem involves a *complete undirected graph* $G = (V, E)$, in which the set V contains all *n* cities, the set E contains all edges of any pair of vertices, and there is a nonnegative cost $c(u, v)$ associated with each edge (u, v) in the set E, and may be modeled as a search of a *Hamiltonian cycle* within *G* with the minimum cost.

The traveling-salesman problem is NP-complete, so it cannot be solved by a polynomial-time algorithm. Many methods have been proposed to speed up the process, such as genetic local search [21], distributed branch-and-bound search [22], annealing-based heuristic search [23], etc. However, the number of the monitoring points in the experiment of this study is not too large. So we adopt an exhaustive search method (or called a brute-force method), whose time complexity is $O(n!)$ with inputs of size n, to find an optimal solution in all combinatorial states.

5.3.2 Proposed technique for generation of complete patrolling paths

Because every monitoring point is visited only once in a session, we transform the path planning proposed into the traveling-salesman problem. Some assumptions are made:

(1) all monitoring points are contained in a set V;

(2) each pair of vertices in the set V has an edge between them and all edges are contained in a set E;

(3) a complete undirected graph G is composed of the set V and the set E; and

(4) a cost $c(u, v)$ associated with the edge (u, v) is the distance between the monitoring points u and v.

If two monitoring points belong to different rectangular regions, the distance between them is calculated by Dijkstra's algorithm described in Section 5.2; else, it is the straight line distance.

As an example, a patrolling environment is shown in Figure 5.10, in which M_1 through M_4 are monitoring points, and N_1 through N_5 are turning points. Furthermore, each black edge connects two points which are in an identical rectangular region and each blue edge connects two monitoring points which are also in the same one. All distances between pairs of monitoring points and turning points passed by them are recorded in a table, as shown in Table 5.1.

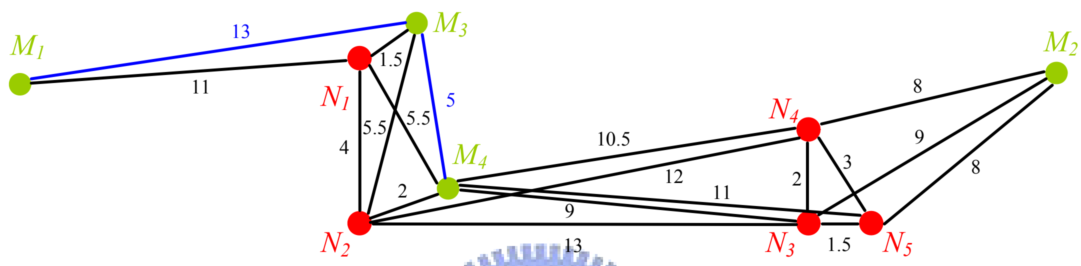


Figure 5.10 A patrolling environment.

Then, we transform the graph in Figure 5.10 into another, as shown in Figure 5.11.With the complete undirected graph G and all costs associated with the edges, we can find an optimal patrolling path. Because the vehicles do not return to the start position in this study, the path is exactly a *Hamiltonian path* with the minimum cost. The result of Figure 5.11 starting at M_1 is $M_1 \rightarrow M_3 \rightarrow M_4 \rightarrow M_2$. And then integrating the information of Table 5.1 into the path, we get the final result as $M_1 \rightarrow M_3 \rightarrow M_4 \rightarrow N_4 \rightarrow M_2$. The algorithm of generating an optimal patrolling path is shown below.

Table 5.1 Distances and passing turning points between every pair of monitoring points.

	M_1	M_2	M_3	M_4
M_1		37	13	16.5

M_2	$N_1 \rightarrow N_2 \rightarrow N_3$ ($M_1 \rightarrow M_2$)		25.5	18.5
M_3		$N_4 \rightarrow N_2$ ($M_2 \rightarrow M_3$)		5
M_4	N_1 ($M_1 \rightarrow M_4$)	N_4 ($M_2 \rightarrow M_4$)		

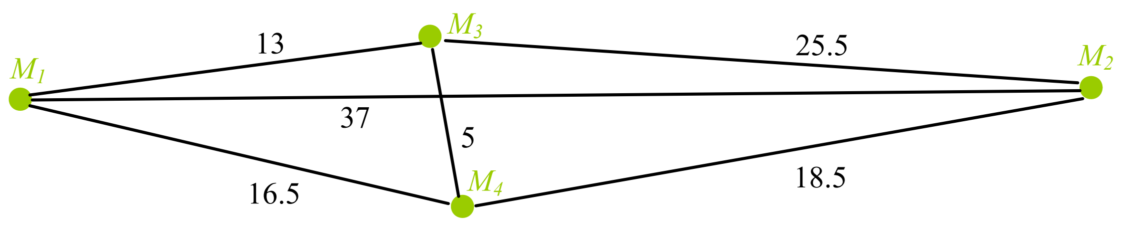


Figure 5.11 A complete undirected graph $G = (V, E)$.

Algorithm 5.3 *Generation of an optimal patrolling path.*

Input: All monitoring points, a table T containing all distances and passing turning points between all pairs of monitoring points, and a position P where the vehicle starts at.

Output: An optimal patrolling path from P.

Steps:

Step 1. Read the distance between each pair of monitoring points from T and each of them is exactly the cost $c(u, v)$ associated with the edge (u, v) .

Step 2. Find a *Hamiltonian path* with the minimum cost from P in the patrolling environment containing all *MP*s.

Step 3. Merge turning points into the *Hamiltonian path* obtained from Step 2, by Table T.

In this study, the number of vehicles used to perform the patrolling task is more than one. Therefore, all monitoring points are divided randomly into groups for each vehicle to patrol. The rule of choosing monitoring points randomly has been described in Section 4.2.1. By performing Algorithm 5.3 individually, the optimal randomized patrolling paths for all vehicles can be obtained. However, we set a threshold parameter *T* to restrict the differences of the patrolling distances for the property of load balancing among vehicles. If the condition is not satisfied, all monitoring points will be chosen and Algorithm 5.3 performed again.

Chapter 6
Collision Avoidance between Vehicles

6.1 Introduction

In this study, an assumption made is that no unexpected obstacle exists in the patrolling environment. However, the security patrolling task is performed by multiple vehicles, it is necessary to ensure no collision among vehicles. Many methods about collision avoidance among multiple vehicles have been proposed, such as by cell decomposition [18], using a probabilistic model [19], or based on multilayered cellular automata architecture [20], etc.

Because only two vehicles are used in the experiment of this study, we can solve the problem by keeping a fixed distance *Dis* between two vehicles. If the distance between the vehicles is smaller than *Dis*, their patrolling paths must be changed. By the way, it is noted that collision avoidance between the vehicles is real-time and this is good for the property of random patrolling paths. Furthermore, the calculating time is short. If the number of the vehicles is more than two, the collision avoidance technique will need more consideration and this can be one of the further works.

6.2 Detection of Collisions

Because vehicles are located constantly by the top-view omni-cameras, the odometer values are credible. Therefore, the values are utilized to compute the distance between two vehicles in every cycle of a fixed time duration. If they are too close, their paths will be changed by inserting some *passing points*. By these points, the distance between the two vehicles can be drawn apart.

For collisions, there are two different states. We only consider the *section* of the assigned patrolling path from the current positions to the goals, turning points or monitoring points, where the vehicles are moving to. If the two sections have an intersection, this state is called *path-intersecting*; else, it is the state of *non-path-intersecting*. The proposed collision avoidance techniques are described in Section 6.3.

6.3 Proposed Collision Avoidance Techniques

6.3.1 Collision avoidance on intersecting paths

In the state of path-intersecting, we let the two vehicles keeping a fixed distance. Assume that the first vehicle is at position $V_{11} = (x_{11}, y_{11})$ and the other is at position $V_{21} = (x_{21}, y_{21})$. Additionally, assume that they are moving to the goals $G_1 = (gx_1, gy_1)$ and $G_2 = (gx_2, gy_2)$, respectively. At first, we calculate the intersection I in the two paths by the *parametric forms*. Each parametric form with respect to the path from the current position to the goal is shown in Eq. (6.1) below:

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} x_{11} \\ y_{11} \end{pmatrix} + r * \begin{pmatrix} gx_1 - x_{11} \\ gy_1 - y_{11} \end{pmatrix} \text{ for vehicle1;}$$
$$\begin{pmatrix} x_2 \\ y_2 \end{pmatrix} = \begin{pmatrix} x_{21} \\ y_{21} \end{pmatrix} + s * \begin{pmatrix} gx_2 - x_{21} \\ gy_2 - y_{21} \end{pmatrix} \text{ for vehicle2;} \qquad (6.1)$$
$$\text{where } 0 \le r, s \le 1.$$

The point I can be obtained by solving the simultaneous equations of "$x_1 = x_2$" and "$y_1 = y_2$". Then, assume that the distance between V_{11} and I is smaller than the distance between V_{21} and I, as shown in Figure 6.1, and so the path of the second vehicle must be changed.

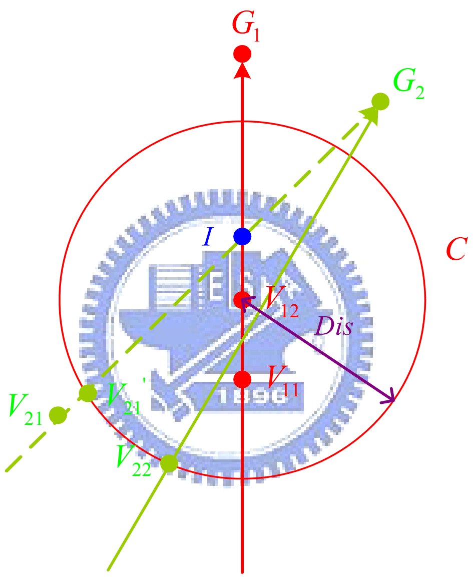


Figure 6.1 An intersection I on the paths of two vehicles.

Because the first vehicle moves along the original path, we can forecast the position $V_{12} = (x_{12}, y_{12})$ at the next moment. If the velocity of the vehicle is *vel* and the time interval of calculating the distance between vehicles is t, then V_{12} can be calculated by Eq. (6.2) below:

$$\begin{pmatrix} x_{12} \\ y_{12} \end{pmatrix} = \begin{pmatrix} x_{11} \\ y_{11} \end{pmatrix} + \frac{vol * t}{\sqrt{(gx_1 - x_{11})^2 + (gy_1 - y_{11})^2}} * \begin{pmatrix} gx_1 - x_{11} \\ gy_1 - y_{11} \end{pmatrix}. \qquad (6.2)$$

Because the vehicles must keep a distance *Dis* to each other, we can say that the passing point V_{22} of the second vehicle is on a circle C whose center is V_{12} and whose radius is *Dis*. To calculate V_{22}, we must acquire the projection point V_{21}' of V_{21} on the circle *C*, at first. The reason is that the distance between V_{21} and V_{22} is desired to be not too long. So we utilize an included angle θ between V_{21}' and V_{22} on *C* to acquire V_{22}, in which *C* and V_{21}' can be represented as follows:

$$C : (x - x_{12})^2 + (y - y_{12})^2 = Dis^2$$
$$V_{21}' = \begin{pmatrix} x_{21}' \\ y_{21}' \end{pmatrix} = \begin{pmatrix} x_{21} \\ y_{21} \end{pmatrix} + s * \begin{pmatrix} gx_2 - x_{21} \\ gy_2 - y_{21} \end{pmatrix}, \text{ where } s \in \Re. \tag{6.3}$$

Therefore, the projection point V_{21}' can be calculated by Eq. (6.4) as follows:

$$(x_{21} + (gx_2 - x_{21})s - x_{12})^2 + (y_{21} + (gx_2 - x_{21})s - y_{12})^2 = Dis^2. \tag{6.4}$$

As long as the value *s* is solved, V_{21}' is obtained. Then, we calculate the angle θ_{21}' of V_{21}' from the positive direction of the *x*-axis. Because *C* can also be represented as Eq. (6.5) below, we can calculate θ_{21}' by Eq. (6.6) below:

$$C : (x_{12} + Dis * \cos\phi, y_{12} + Dis * \sin\phi); \tag{6.5}$$

$$x_{21}' = x_{12} + Dis * \cos\theta_{21}'$$
$$\text{where } \theta_{21}' = arc \ \cos(\frac{x_{21}' - x_{12}}{Dis}). \tag{6.6}$$

Because the included angle between V_{21}' and V_{22} is restricted to be θ, V_{22} is one of the points *a* or *b* as shown in Eq. (6.7), called *candidate passing points*:

$$a = \begin{pmatrix} x_{12} \\ y_{12} \end{pmatrix} + Dis * \begin{pmatrix} \cos(\theta_{21}' + \theta) \\ \sin(\theta_{21}' + \theta) \end{pmatrix}$$
$$b = \begin{pmatrix} x_{12} \\ y_{12} \end{pmatrix} + Dis * \begin{pmatrix} \cos(\theta_{21}' - \theta) \\ \sin(\theta_{21}' - \theta) \end{pmatrix} \tag{6.7}$$

Furthermore, it is desired that the distance between V_{22} and G_1 is bigger. The reason is that the distance between the two vehicles can be drawn apart. Therefore, if point a is closer to G_1 than point b, V_{22} will be exactly the point b. As the passing point V_{22} is obtained, the patrolling path of the second vehicle is changed from $V_{21} \rightarrow G_2$ to $V_{21} \rightarrow V_{22} \rightarrow G_2$.

It is possible that the alternative path may exceed the walkable range. If this is the case, then the path must be changed again. In general, there are three unallowable states:

(1) the passing point P is in the outer region as shown in Figure 6.2;

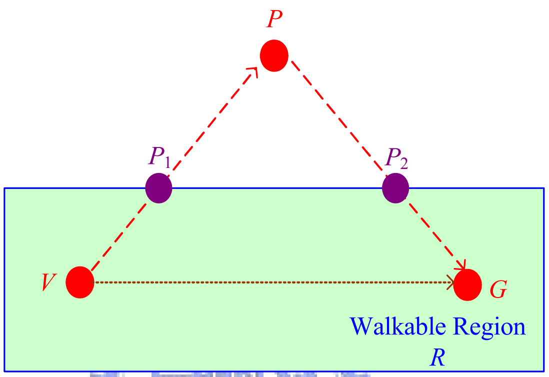


Figure 6.2 Passing point P is in the outer region.

(2) the path from the current position V to the passing point P exceeds the walkable range as shown in Figure 6.3; and

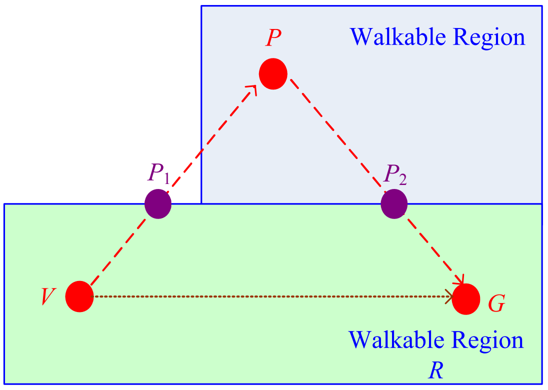


Figure 6.3 The alternative path is not feasible.

(3) the path from the passing point P to the goal G exceeds the walkable range as shown in Figure 6.4.

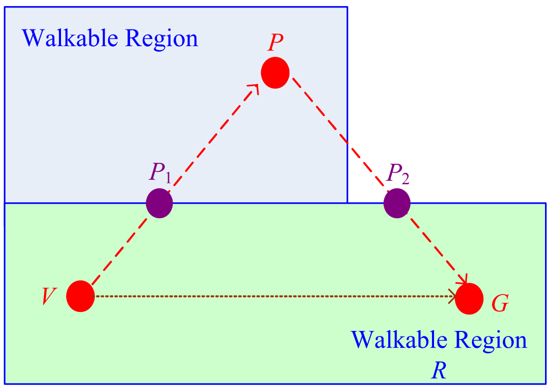


Figure 6.4 The alternative path is not feasible.

For these states, the rectangular region which the passing point P belongs to is not equal to the one R which both V and G belong to. Therefore, we calculate the intersections of the region R and the alternative path $V \rightarrow P \rightarrow G$. Finally, the patrolling path is taken to be $V \rightarrow P_1 \rightarrow P_2 \rightarrow G$. The algorithm of checking and finding a feasible alternative patrolling path is shown in the following.

Algorithm 6.1 *Checking and finding a feasible alternative patrolling path.*

Input: All rectangular regions Re, the passing point P, the front point F (the current position V or another passing point) of P, and the goal G.

Output: A feasible alternative patrolling path.

Steps:

Step 1. Find the rectangular region R, which both F and G belong to, from Re.

Step 2. Check whether the rectangular region, which the passing point belongs to, is identical to R.

If yes, the alternative patrolling path is $F \rightarrow P \rightarrow G$.

Else, go to Step 3.

Step 3. Calculate the intersections, P_1 and P_2, of the region R and the alternative path $F \rightarrow P \rightarrow G$.

Then, the alternative patrolling path is $F \rightarrow P_1 \rightarrow P_2 \rightarrow G$.

Additionally, after the time interval t, the system must check whether an intersection is still on the paths of the two vehicles again. If yes, the patrolling path of the second vehicle will be changed again. The entire process is described as Algorithm 6.2.

Algorithm 6.2 *Computing an alternative path for an intersecting state.*

Input: The current position $V_{11} = (x_{11}, y_{11})$ and the goal $G_1 = (gx_1, gy_1)$ of the first vehicle, the current position $V_{21} = (x_{21}, y_{21})$ and the goal $G_2 = (gx_2, gy_2)$ of the second vehicle, a fixed length *Dis* between the two vehicles, the velocity *vel* of the vehicles, the time interval t of calculating the distance between the vehicles, and the included angle θ between the current position and the passing point.

Output: A alternative patrolling path for one vehicle.

Steps:

Step 1. Calculate the intersection I in the two paths $V_{11} \rightarrow G_1$ and $V_{21} \rightarrow G_2$.

Step 2. Judge which vehicle should change its path.

If $d(V_{11}, I) < d(V_{21}, I)$, the path of the second vehicle must be changed.

Else, the path of the first vehicle must be changed.

Assume that the path of the second vehicle must be changed in the following steps.

Step 3. Forecast the position V_{12} of V_{11} at the next moment by the following equation:

$$\begin{pmatrix} x_{12} \\ y_{12} \end{pmatrix} = \begin{pmatrix} x_{11} \\ y_{11} \end{pmatrix} + \frac{vol * t}{\sqrt{(gx_1 - x_{11})^2 + (gy_1 - y_{11})^2}} * \begin{pmatrix} gx_1 - x_{11} \\ gy_1 - y_{11} \end{pmatrix}.$$

Step 4. Calculate the projection point V_{21}' of V_{21} on the circle C whose center is

V_{12} and whose radius is *Dis* by the following equation:

$$C:(x - x_{12})^2 + (y - y_{12})^2 = Dis^2$$
$$V_{21}' = \begin{pmatrix} x_{21}' \\ y_{21}' \end{pmatrix} = \begin{pmatrix} x_{21} \\ y_{21} \end{pmatrix} + s * \begin{pmatrix} gx_2 - x_{21} \\ gy_2 - y_{21} \end{pmatrix}, \text{ where } s \in \Re.$$

The above equations lead to

$$(x_{21} + (gx_2 - x_{21})s - x_{12})^2 + (y_{21} + (gy_2 - y_{21})s - y_{12})^2 = Dis^2 .$$

Step 5. Calculate the angle θ_{21}' of V_{21}' from the positive direction of the x-axis by the following equation:

$$C:(x_{12} + Dis * \cos\phi, y_{12} + Dis * \sin\phi);$$
$$V_{21}' = (x_{21}', y_{21}').$$

The above equations lead to:

$$x_{21}' = x_{12} + Dis * \cos\theta_{21}';$$
$$\theta_{21}' = arc\ \cos(\frac{x_{21}' - x_{12}}{Dis}).$$

Step 6. Calculate candidate passing points *a* and *b* in the following way, in which the included angle between V_{21}' and the candidate passing points is θ:

$$a = \begin{pmatrix} x_{12} \\ y_{12} \end{pmatrix} + Dis \begin{pmatrix} \cos(\theta_{21}' + \theta) \\ \sin(\theta_{21}' + \theta) \end{pmatrix};$$
$$b = \begin{pmatrix} x_{12} \\ y_{12} \end{pmatrix} + Dis \begin{pmatrix} \cos(\theta_{21}' - \theta) \\ \sin(\theta_{21}' - \theta) \end{pmatrix}.$$

Step 7. Determine the passing point in the following way:

if $(a, G_1) < (b, G_1)$, point *b* is the passing point.

Then the path of the second vehicle is taken to be $V_{21} \rightarrow b \rightarrow G_2$.

Step 8. Check whether the alternative path is feasible by Algorithm 6.1.

If not, the new alternative path will be obtained from Algorithm 6.1.

6.3.2 Collision avoidance on non-intersecting paths

In this state, both paths of the two vehicles must be changed. Each passing point is on the perpendicular bisector l of the path from a current position $V = (x_1, y_1)$ to a goal $G = (x_2, y_2)$. Because it is desired to draw apart the distance between the two vehicles quickly, the included angle between the paths $V \rightarrow G$ and $V \rightarrow$ turning point $P = (p_1, p_2)$ is set by a parameter θ. Therefore, the distance between P and C, which is the midpoint of line segment V and G, is:

$$\frac{\overline{VG}}{2} * \tan\theta . \tag{6.8}$$

A graphic illustration is shown in Figure 6.5.

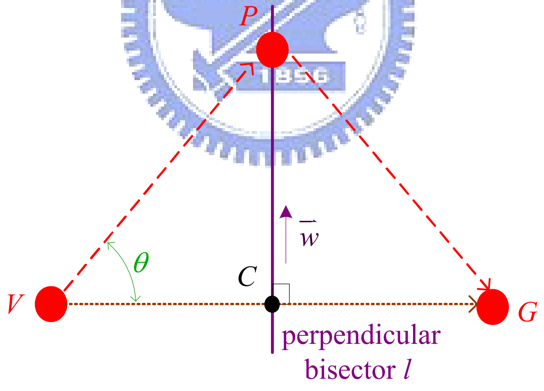


Figure 6.5 An included angle θ between $V \rightarrow G$ and $V \rightarrow P$.

The directional vector $\vec{w}$ of l is perpendicular to the one of path $V \rightarrow G$, so it is:

$$\vec{w} = \begin{pmatrix} -(y_1 - y_2) \\ x_1 - x_2 \end{pmatrix}. \tag{6.9}$$

Then, l can be represented as:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \frac{1}{2}\begin{pmatrix} x_1 + x_2 \\ y_1 + y_2 \end{pmatrix} + s * \begin{pmatrix} -(y_1 - y_2) \\ (x_1 - x_2) \end{pmatrix}, \tag{6.10}$$

where $s \in \Re$.

Because P is on the perpendicular bisector l and the distance between C and P is computed by Eq. (6.8), P can be represented by Eq. (6.11) below, in which a and b are in the different sides of path $V \rightarrow G$, respectively:

$$a = \frac{1}{2}\begin{pmatrix} x_1 + x_2 \\ y_1 + y_2 \end{pmatrix} - \frac{\overline{VG} * \tan\theta}{2 * \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}} \begin{pmatrix} -(y_1 - y_2) \\ x_1 - x_2 \end{pmatrix};$$

or (6.11)

$$b = \frac{1}{2}\begin{pmatrix} x_1 + x_2 \\ y_1 + y_2 \end{pmatrix} + \frac{\overline{VG} * \tan\theta}{2 * \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}} \begin{pmatrix} -(y_1 - y_2) \\ x_1 - x_2 \end{pmatrix}.$$

For each path of the vehicles, there are two candidates to choose as a passing point. The condition of choosing passing points for the vehicles is that the distance between chosen passing points is the longest within all combinations of the candidates. See an example shown in the following.

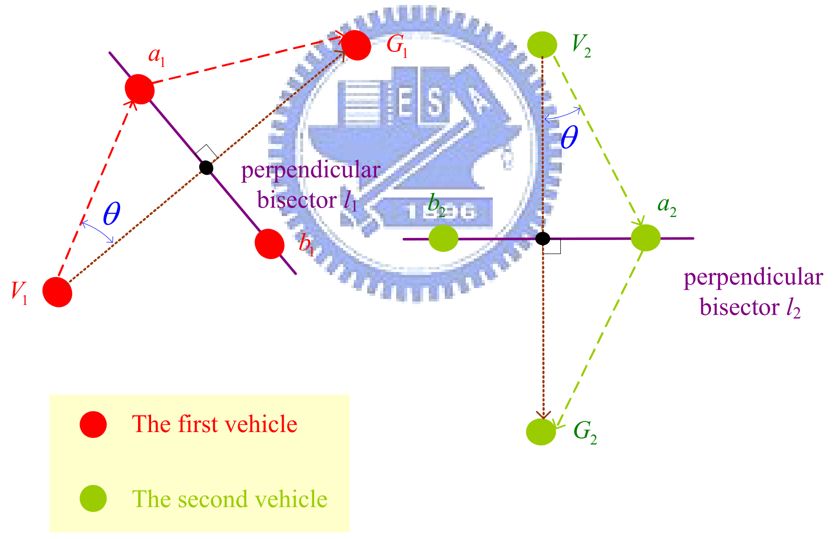


Figure 6.6 Alternative paths at a non-intersecting state.

Points a_1 and b_1 are the candidates for the first vehicle and points a_2 and b_2 are the ones for the second vehicle. All states of choosing candidates are (a_1, a_2), (a_1, b_2), (b_1, a_2), and (b_1, b_2), in which the distance between a_1 and a_2 is the shortest. So, a_1 and a_2 are the passing points for the first and the second vehicles, respectively; the alternative paths for the two vehicles are $V_1 \rightarrow a_1 \rightarrow G_1$ and $V_2 \rightarrow a_2 \rightarrow G_2$.

Additionally, because the included angle between the two paths from the current position V to the goal G and to the passing point P is restricted by a parameter θ, the distance between P and path $V \rightarrow G$ may be too long. Therefore, we set a fixed distance D between them. If the distance between P and path $V \rightarrow G$ is larger than D, we translate the line segment l connecting V and G along a perpendicular direction toward the passing point P, such as shown in Figure 6.7, in which l is exactly the translation line.

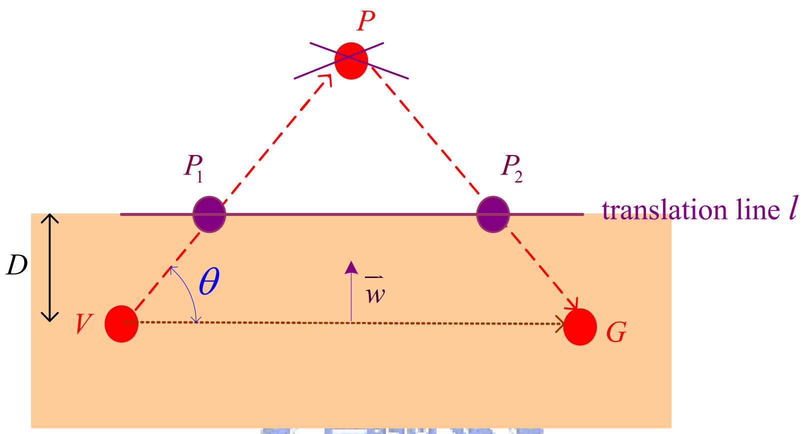


Figure 6.7 The passing point P is too far.

The *translation line l* can be represented as:

$$\begin{aligned}&(y_1 - y_2) * x - (x_1 - x_2) * y = D * \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} + d \\ &\text{or} \\ &(y_1 - y_2) * x - (x_1 - x_2) * y = -D * \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} + d \\ &\text{, if } \overleftrightarrow{VG}:\ (y_1 - y_2) * x - (x_1 - x_2) * y = d.\end{aligned} \tag{6.12}$$

Then, we calculate the intersections, P_1 and P_2, of path $V \rightarrow P \rightarrow G$ and line l. Therefore, the new alternative path is $V \rightarrow P_1 \rightarrow P_2 \rightarrow G$. Furthermore, it is necessary to check whether the path is feasible by Algorithm 6.1, and then the final alternative path can be obtained. See an example shown in Figure 6.8 through Figure 6.10 for illustration.

In the example, two vehicles are too close and paths of them are non-intersecting. Here, we only see the state of one vehicle, as shown in Figure 6.8. P is the passing point, but the point is too far from line segment $\overline{VG}$. Therefore, we translate $\overline{VG}$ a distance D toward P, and then the translation line l can be obtained, as shown in Figure 6.9.

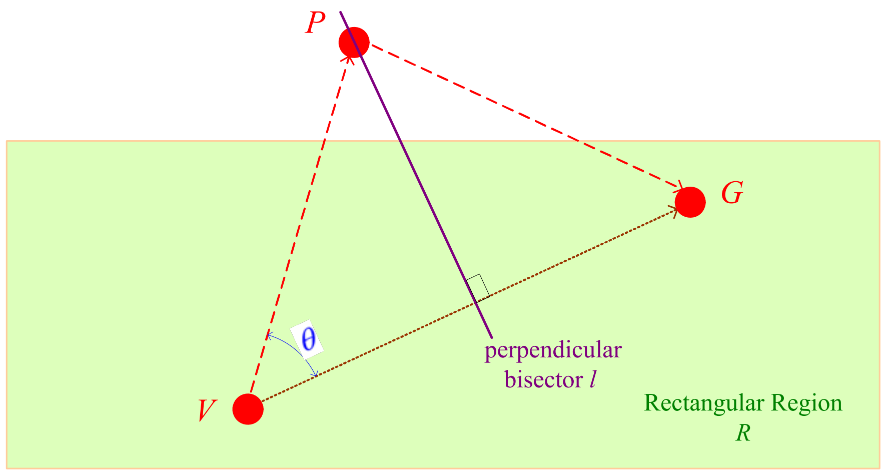


Figure 6.8 A passing point P at a non-intersecting state.

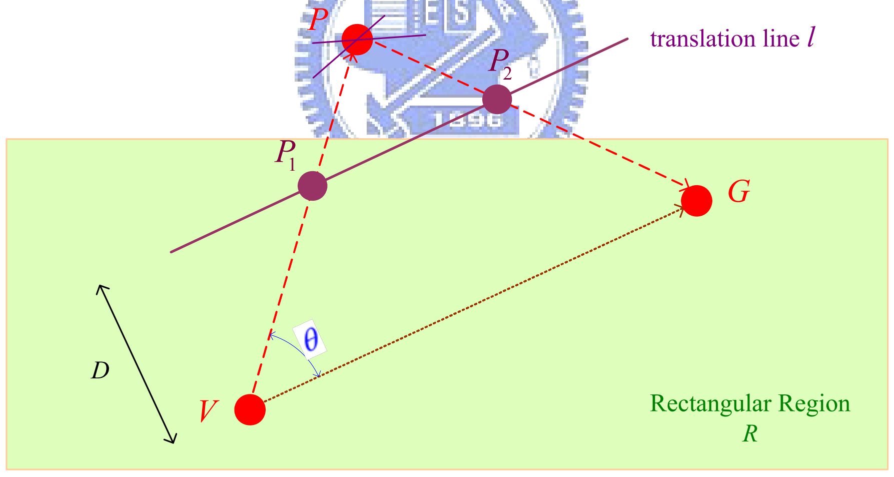


Figure 6.9 P is too far.

After calculating two intersections of l and $V \rightarrow P \rightarrow G$, the new alternative path is $V \rightarrow P_1 \rightarrow P_2 \rightarrow G$. For P_1 and P_2, it is necessary to check whether they are feasible by Algorithm 6.1. Because the rectangular region which P_2 belongs to is different from the one R which both the current position V and the goal G belong to, as shown in Figure 6.10.

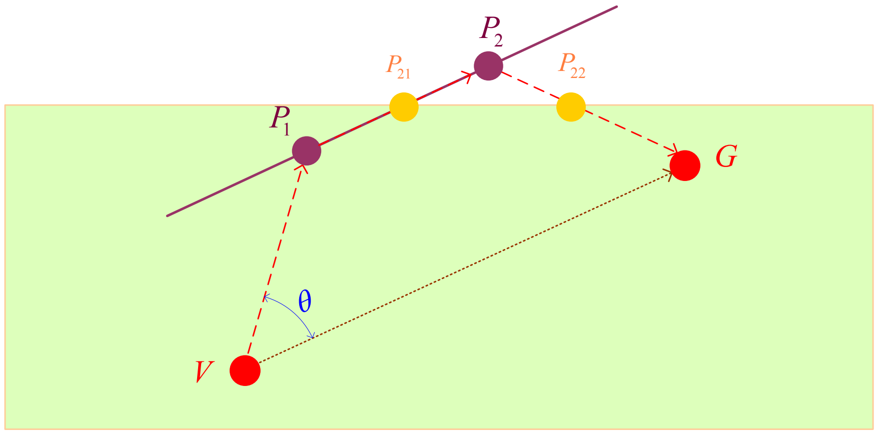


Figure 6.10 P_2 is not in the rectangular region R.

Hence, we calculate the intersections of $P_1 \rightarrow P_2 \rightarrow G$ and R, and then the alternative path becomes $V \rightarrow P_1 \rightarrow P_{21} \rightarrow P_{22} \rightarrow G$. The processing of computing a alternative path at a non-intersecting state is shown as Algorithm 6.3 below.

Algorithm 6.3 *Computing an alternative path for a non-intersecting case.*

Input: The current position $V_1 = (x_{11}, y_{11})$ and the goal $G_1 = (x_{12}, y_{12})$ of the first vehicle, the current position $V_2 = (x_{21}, y_{21})$ and the goal $G_2 = (x_{22}, y_{22})$ of the second vehicle, the restricted distance D between a passing point and the original path $V_1 \rightarrow G_1$ or $V_2 \rightarrow G_2$, and the included angle θ between the original path and the path from the current position to the passing point.

Output: The alternative paths for the two vehicles.

Steps:

Step 1. Calculate two candidates of passing points for the first vehicle (a_1 and b_1) and the second vehicle (a_2 and b_2) by the following equation:

$$a_1 = \frac{1}{2}\begin{pmatrix} x_{11}+x_{12} \\ y_{11}+y_{12} \end{pmatrix} - \frac{\overline{VG} * \tan\theta}{2*\sqrt{(x_{11}-x_{12})^2+(y_{11}-y_{12})^2}}\begin{pmatrix} -(y_{11}-y_{12}) \\ x_{11}-x_{12} \end{pmatrix}$$
$$b_1 = \frac{1}{2}\begin{pmatrix} x_{11}+x_{12} \\ y_{11}+y_{12} \end{pmatrix} + \frac{\overline{VG} * \tan\theta}{2*\sqrt{(x_{11}-x_{12})^2+(y_{11}-y_{12})^2}}\begin{pmatrix} -(y_{11}-y_{12}) \\ x_{11}-x_{12} \end{pmatrix}$$
$$a_2 = \frac{1}{2}\begin{pmatrix} x_{21}+x_{22} \\ y_{21}+y_{22} \end{pmatrix} - \frac{\overline{VG} * \tan\theta}{2*\sqrt{(x_{21}-x_{22})^2+(y_{21}-y_{22})^2}}\begin{pmatrix} -(y_{21}-y_{22}) \\ x_{21}-x_{22} \end{pmatrix}$$
$$b_2 = \frac{1}{2}\begin{pmatrix} x_{21}+x_{22} \\ y_{21}+y_{22} \end{pmatrix} + \frac{\overline{VG} * \tan\theta}{2*\sqrt{(x_{21}-x_{22})^2+(y_{21}-y_{22})^2}}\begin{pmatrix} -(y_{21}-y_{22}) \\ x_{21}-x_{22} \end{pmatrix}$$

Step 2. Determine the passing points for the two vehicles in the following way.

Step 2.1. Calculate all distances of the four combinations (a_1, a_2), (a_1, b_2), (b_1, a_2), and (b_1, b_2).

Step 2.2. Choose the least distance from Step 2.1 and the combination is exactly the passing points for the two vehicles.

Step 2.3. Insert the passing point into the original path for each vehicle.

Step 3. Check whether the distance between the passing point P and the original path is larger than D, for each vehicle.

If yes, calculate the intersections of the translation line l and the alternative path, in which the distance between l and the original path is D; and then, replace P by the intersections.

Step 4. Check whether every alternative path is feasible by Algorithm 6.1.

If not, obtain the new alternative path from Algorithm 6.1.

Chapter 7
Experimental Results and Discussions

In this chapter, we show some experimental results of the proposed security patrolling system by two ways. The first is the result of optimal randomized patrolling paths and collision avoidance between vehicles shown by a simulation using programs written in the Borland C++ builder; it is described in Section 7.1. The other is the result of an actual environment in the Computer Vision Laboratory, Department of Computer Science, National Chiao Tung University, and it is described in Section 7.2.

7.1 Experimental Results of Simulation of Patrolling

In this simulation, we create a patrolling environment whose floor shape is composed of four rectangular regions, as shown in Figure 7.1, in which *Obj*. 0 through *Obj*.6 are monitoring points.

We utilize the rectangular regions to calculate all the turning points and the distance between each pair of monitoring points, and then save the data into some text files. By reading the files, they can be used again. Furthermore, the threshold to restrict the differences between the patrolling distances needs to be keyed in to the user interface, as shown in Figure 7.1, marked by a red frame. All patrolling paths are random, optimal, and load balanced among vehicles. See an example shown in Figure 7.2, in which the first vehicle starts its navigation at *Obj*. 4 (*M*4) and the second vehicle starts at *Obj*. 1 (*M*1).

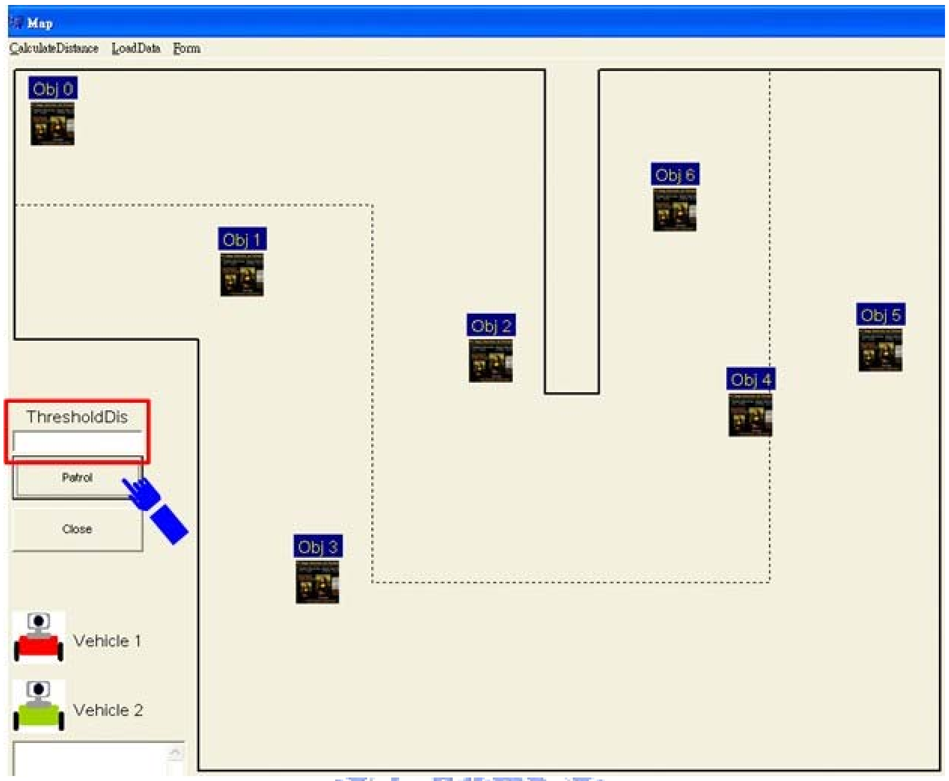
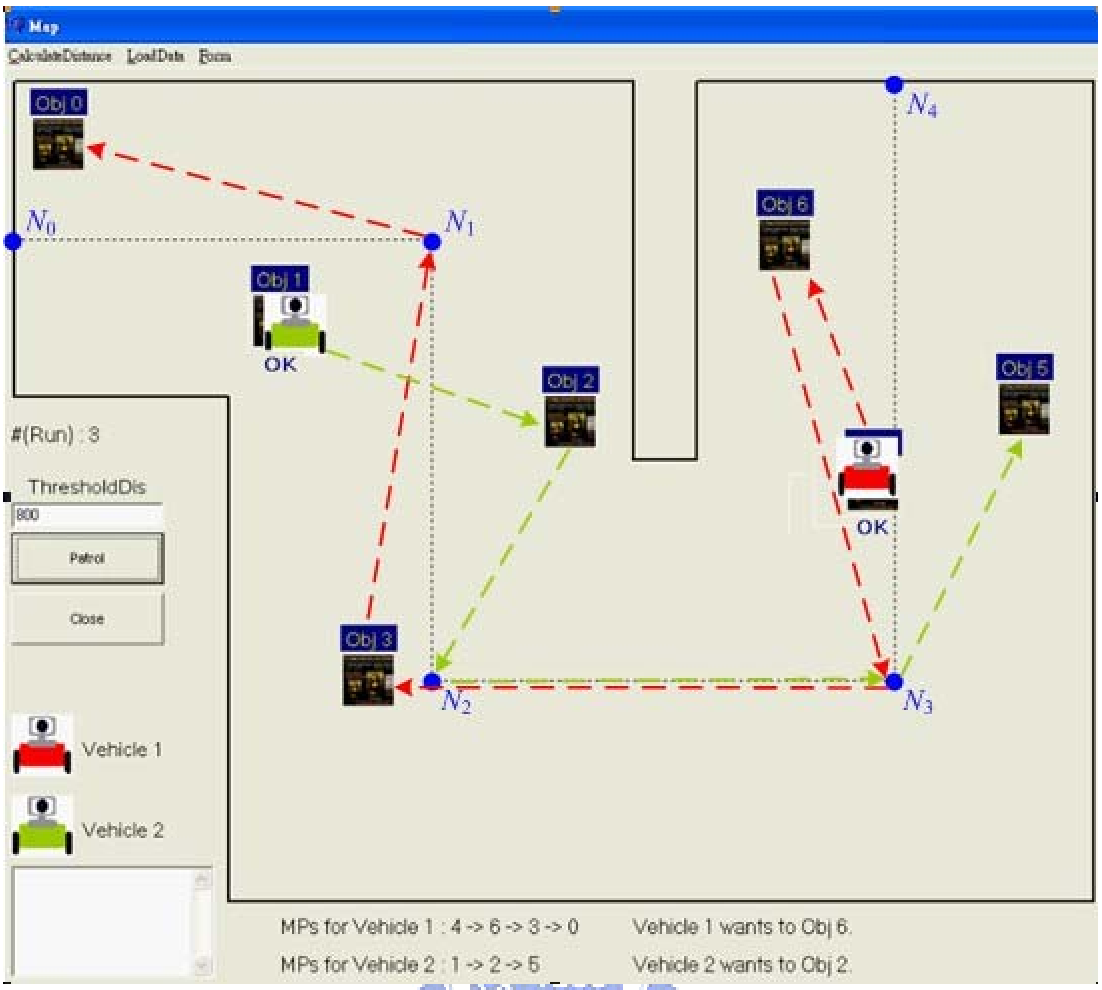


Figure 7.1 A simulated patrolling environment.

Among all monitoring points, *M*0, *M*3, *M*4, and *M*6 are chosen by the first vehicle; *M*1, *M*2, and *M*5 are chosen by the second. The obtained optimal paths are *M*4 → *M*6 → *M*3 → *M*0 and *M*1 →*M*2 → *M*5. According to the record of turning points passed by between each pair of monitoring points, obtained in the learning phase, the actual paths are *M*4 → *M*6 → *N*3 → *M*3 → *N*1 → *M*0 and *M*1 →*M*2 → *N*2 → *N*3→ *M*5 for the two vehicles, as shown by red and green dotted lines in Figure 7.2. Furthermore, the distances of the paths are 1633.22 and 1081.56. Because the difference of the distances is smaller than the threshold 800, set by the user, the two paths are accepted. In this session, the two vehicles end at *M*0 and *M*5, respectively, so the positions are the starts for them in the next patrolling session, for which, the new path planning is shown in Figure 7.3.

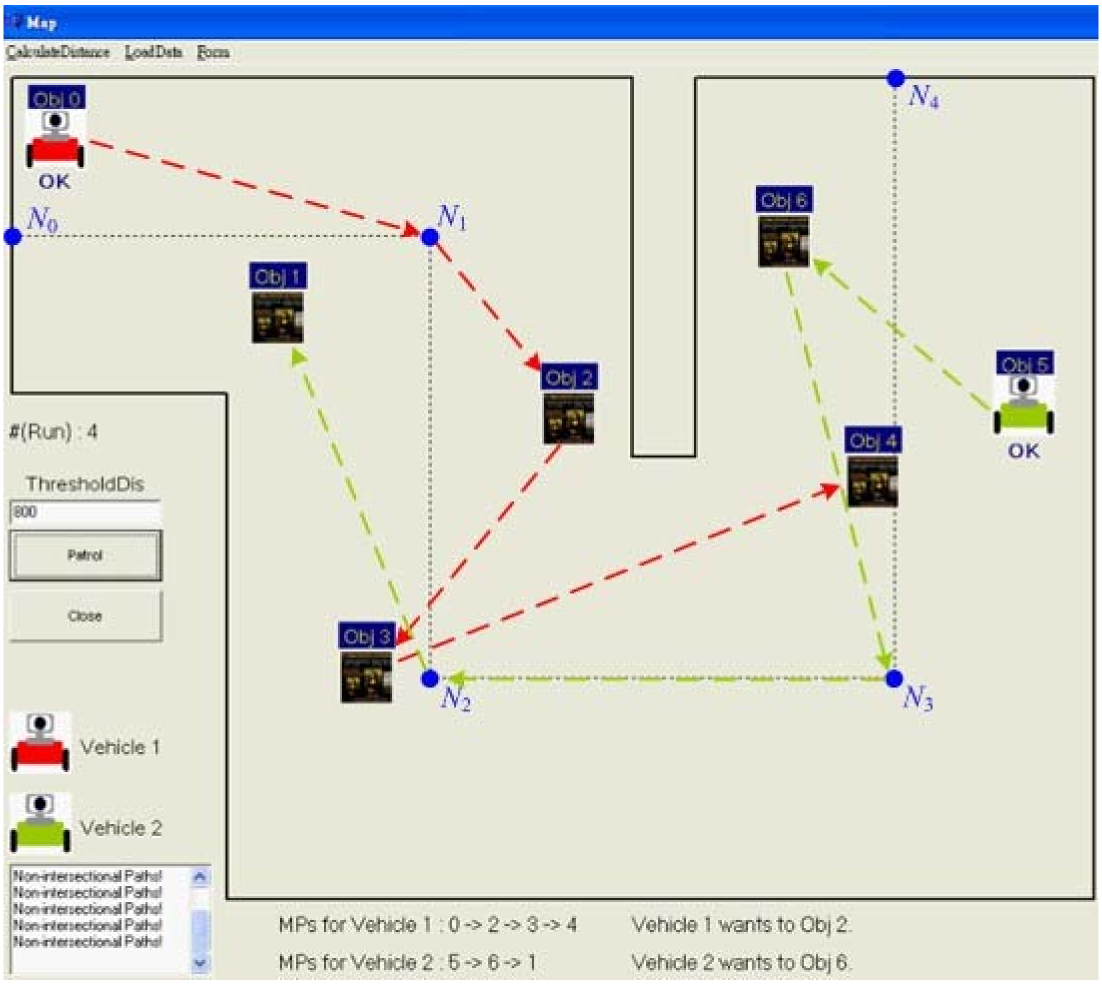


MPs for Vehicle 1 : 4 -> 6 -> 3 -> 0
Vehicle 1 wants to Obj 6.
MPs for Vehicle 2 : 1 -> 2 -> 5
Vehicle 2 wants to Obj 2.

Figure 7.2 Path planning for the two vehicles in a session.

*M*0, *M*2, *M*3, and *M*4 are chosen by the first vehicle; *M*1, *M*5 and *M*6 are chosen by the second. The obtained optimal paths are *M*0 → *M*2 → *M*3 → *M*4 and *M*5 →*M*6 → *M*1; the actual paths are *M*0 → *N*1 → *M*2 → *M*3 → *M*4 and *M*5 →*M*6 → *N*3 → *N*2 → *M*1 for the two vehicles, as shown by the red and green dotted lines in Figure 7.3. Furthermore, the distances of the paths are 1175.41 and 1262.25 and they are also accepted.

To show the advantage of our system, we compare the times needed for different control factors, as shown in Table 7.1. If the property of randomization is an essential condition, the average time in one session taken by using one vehicle is nearly double of that taken by using two vehicles. This result tells us that the system for multiple vehicles can bring more benefits. Besides, if the number of vehicles is the same, an optimal patrolling path will take less time than a non-optimal path.



#(Run) : 4

800

Non-intersectional Paths!
Non-intersectional Paths!
Non-intersectional Paths!
Non-intersectional Paths!
Non-intersectional Paths!

MPs for Vehicle 1 : 0 -> 2 -> 3 -> 4 Vehicle 1 wants to Obj 2.

MPs for Vehicle 2 : 5 -> 6 -> 1 Vehicle 2 wants to Obj 6.

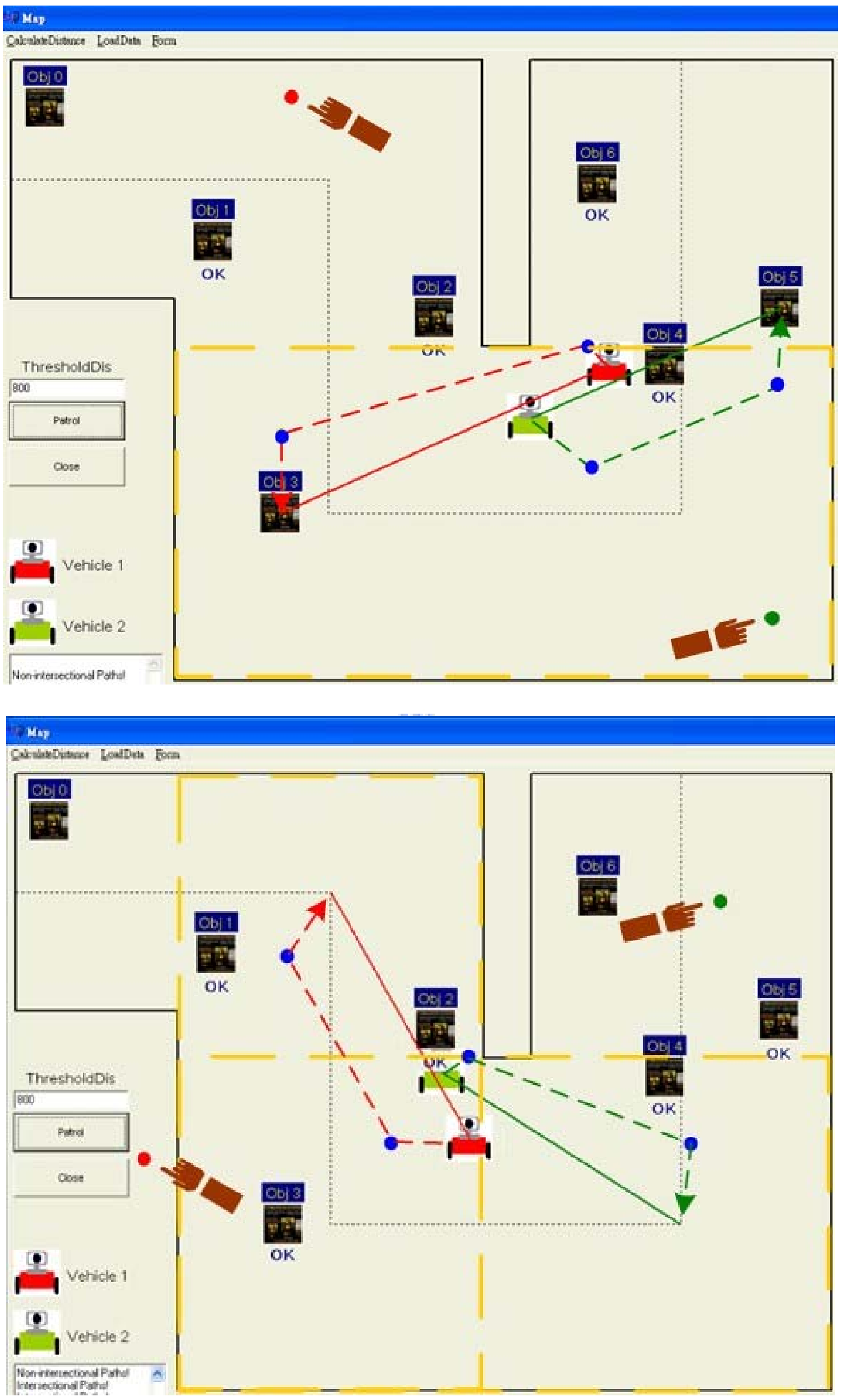
Figure 7.3 Path planning for the two vehicles in the next session.

For collision avoidance, if the distance between two vehicles is too close, the paths of the vehicles will be changed. The states of non-intersecting paths are shown in Figure 7.4, in which red and green lines are the original paths of the vehicles. Because the first obtained passing points, red and green circles, are too far from the original path or exceed the walkable regions, the blue circles are calculated. The dotted lines are exactly the feasible alternative paths for the vehicles. Besides, the final passing points belong to the rectangular region which the original path also belongs to.

Table 7.1 The table of time comparisons where O and X means conducted or not, respectively.

Number of Vehicles	**Randomization**	**Optimization**	**Average Time (second / one session)**	**Saved Time / Original Time (%)**
1	**O**	**O**	**39.4**	**-**
1	**O**	**X**	**31.6**	**19.8**
2	**O**	**O**	**19.7**	**50.0**
2	**O**	**X**	**13.5**	**65.7**

For collision avoidance of the path-intersecting case, one example is shown in Figure 7.5. Because the second vehicle is closer to the intersection than the first one, the path of the first vehicle must be changed. The black circles *a* and *b* are the candidates of passing points. Because *b* is farther from the goal of the second vehicle, *b* is chosen. However, the rectangular region which *b* belongs to is different from the one which the original path belongs to. The blue circle is the final passing point to be chosen, and the red dotted line is the alternative path for the first vehicle.



Non-intersectional Path!
Intersectional Path!

Figure 7.4 Collision avoidance of non-intersecting paths.

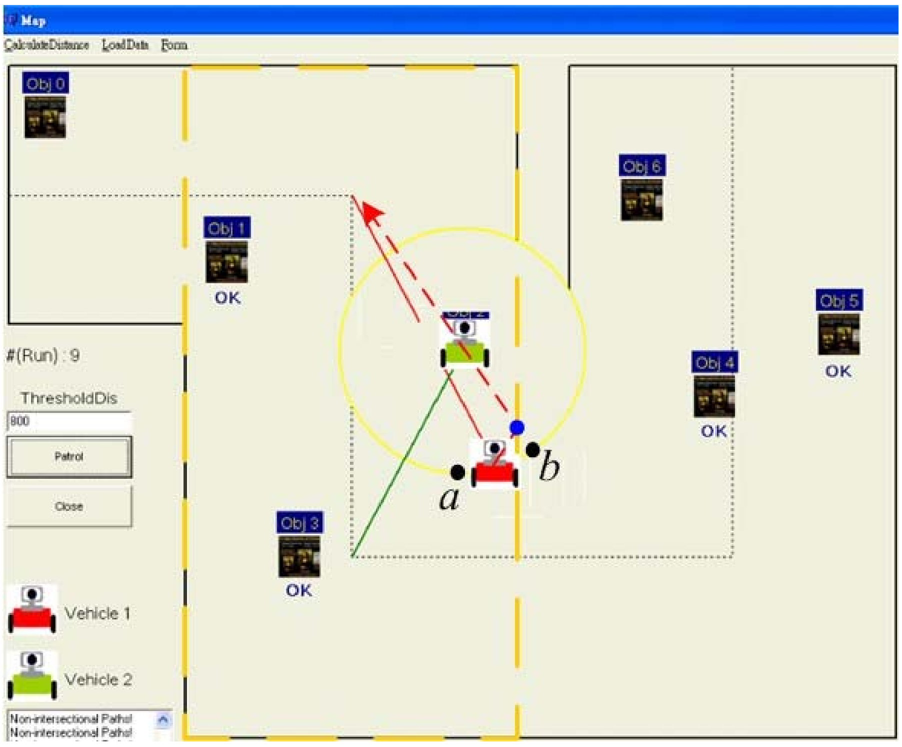


Figure 7.5 Collision avoidance of intersecting paths.

7.2 Experimental Results of Patrolling in Real Environment

The real environment for this experiment is an open space area in our laboratory. Because autonomous vehicles used in the study suffer from accumulation of mechanical errors, two top-view omni-cameras are utilized to locate and monitor the vehicles.

While collecting data for Table 7.2, we drive the vehicle to random places and record the values of the actual positions and the odometer. The total moved distance, passing twenty position points, is 4818.40 centimeters and the average error rate without calibration by tow-view cameras between the actual positions and the odometer values is 8.86%. Furthermore, the reason why the errors do not increase is that turning of the vehicle also incurs errors, so the errors might cancel one another by left and right turnings.

Table 7.2 Mechanical errors of the vehicle.

No.	(1)Actual Position		(2)Odometer Value		Error $(\frac{\lvert(1)-(2)\rvert}{(1)})$
	x	y	x	y	
1	47.6	7.4	45.9	7.4	0.035
2	163.4	32.2	160.7	34.2	0.020
3	272.3	35.2	269.3	41.1	0.024
4	382.2	29.8	378.8	40.9	0.030
5	399	85.3	392.8	97.3	0.033
6	304.5	115.5	299.2	124.2	0.031
7	546.6	52	546.9	79.8	0.051
8	799.9	145.2	786.6	197.9	0.067
9	505.1	204.3	491.6	228	0.050
10	267.4	198.2	256.8	192.8	0.036
11	717.3	195.2	691.8	274.6	0.112
12	353.8	193.5	335.9	201.8	0.049
13	789.8	37.2	791.2	138.2	0.128
14	573.5	139.7	557.4	175	0.066
15	631.6	82	630.9	138.9	0.089
16	329.7	177.6	316.2	104.5	0.198
17	746.6	103.2	721.9	208.6	0.144
18	213	46	292.9	106.5	0.460
19	635.8	203.3	574.9	243.8	0.110
20	523	97.4	543	94	0.038

Additionally, we also record the errors of the two top-view omni-cameras. The image in Figure 7.6 is the view of the first camera and we calculate the positions of all circles in the image.

In Table 7.3, we calculate the errors between the actual positions and the positions in the image. From the values, we know that the errors of those points, which is farther from the view center of the camera, is bigger. However, the average error rate with calibration is only 3.86% and all points are independent. Therefore, the vehicles are located by the cameras such that the vehicles do not suffer from accumulation of mechanical errors anymore. About the second camera, the view and the errors are shown in Figure 7.7 and Table 7.4, in which the average error rate with calibration is 2.51 %.

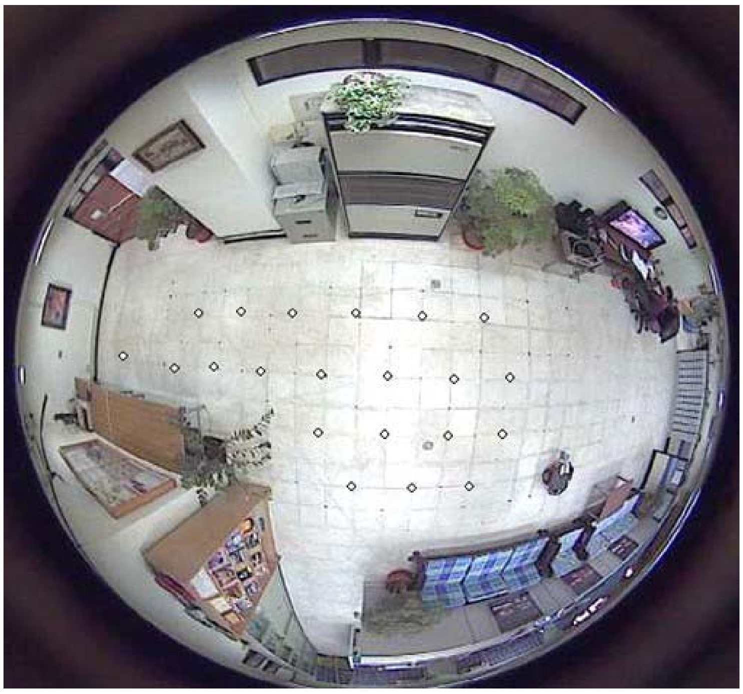


Figure 7.6 The view of the first top-view omni-camera.

Furthermore, the task of security patrolling includes the work of capturing the pictures of some monitored objects. By the top-view omni-cameras to locate the vehicles periodically in the patrolling session, the vehicles can accomplish the mission with the information of the positions and the orientations with respect to the objects, obtained in the learning phase. In the following, we show some results of images taken by the vehicles. Some monitored objects are in the center of the images, as shown in Figure 7.8(a). The images, captured by the vehicles in the patrolling session, with respect to the ones in Figure 7.8(a) are shown in Figure 7.8(b).

Table 7.3 Errors of the first top-view omni-camera.

No.	(1)Actual Position		(2)Image Position		Error $(\frac{\lvert(1)-(2)\rvert}{(1)}\%)$
	X	y	x	y	
1	61	30.5	52.89	28.56	0.122
2	122	30.5	113.62	28.68	0.068
3	183	30.5	174.53	30.5	0.046
4	244	30.5	238.19	30.5	0.024
5	305	30.5	303.48	30.58	0.005
6	366	30.5	368.11	27.17	0.011
7	30.5	91.5	15.25	99.13	0.177
8	91.5	91.5	80.61	93.68	0.086
9	152.5	91.5	141.74	95.15	0.064
10	213.5	91.5	207.4	93.43	0.028
11	274.5	91.5	271.6	91.65	0.010
12	335.5	91.5	337.19	93.19	0.007
13	396.5	91.5	400.31	91.5	0.009
14	213.5	152.5	205.47	156.39	0.034
15	274.5	152.5	270.22	154.21	0.015
16	335.5	152.5	335.86	155.89	0.009
17	396.5	152.5	397.27	156.36	0.009
18	244	213.5	239.39	217.91	0.020
19	305	213.5	302.22	217.84	0.014
20	366	213.5	367.24	220.12	0.016

In Figure 7.8, the difference between each pair of images is smaller. It tells us that the proposed vehicle-pose learning strategy and the proposed vehicle localization technique are good for the vehicles to perform the security patrolling task.

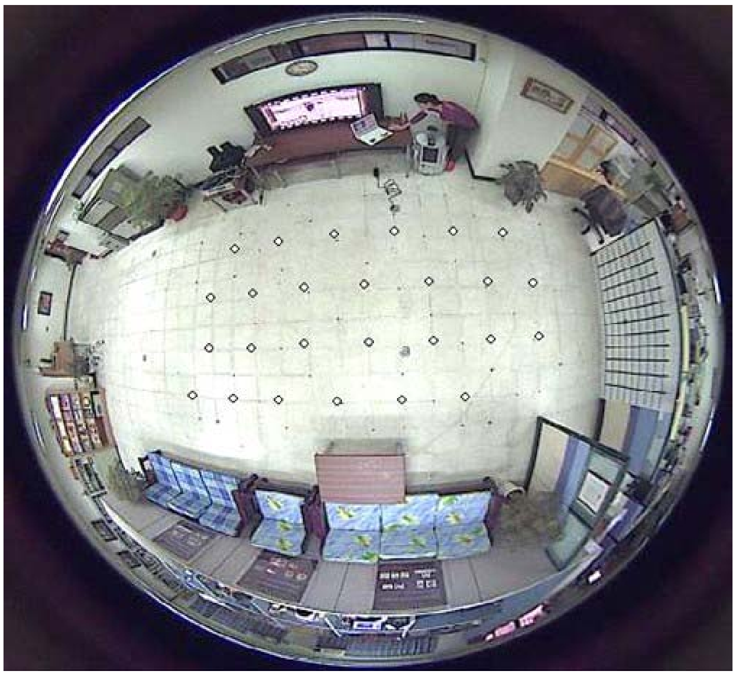
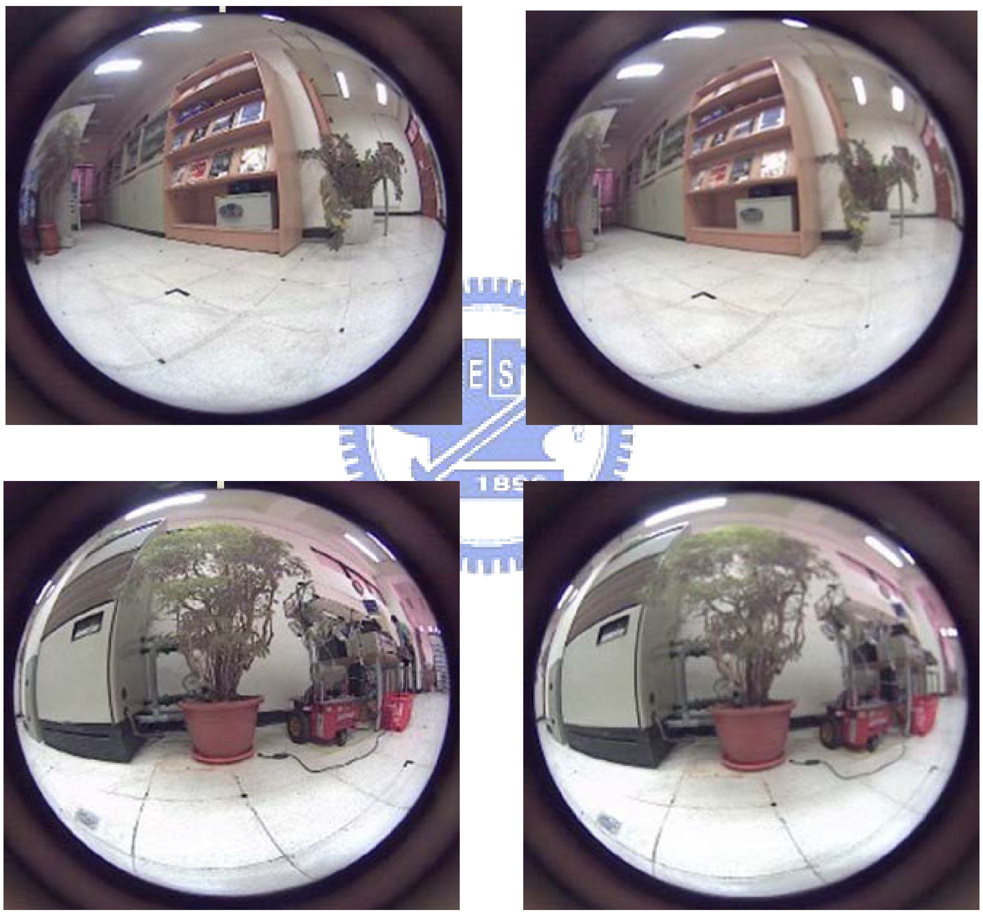


Figure 7.7 The view of the second top-view omni-camera.

Table 7.4 Errors of the second top-view omni-camera.

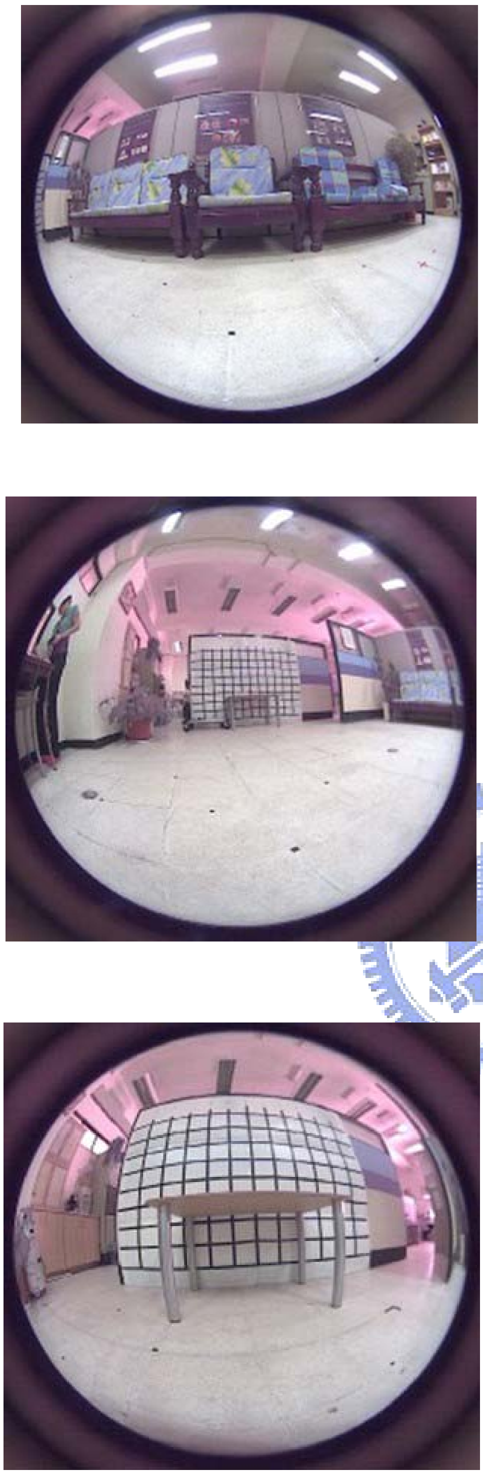
No.	Actual Position		Image Position		Error $(\frac{\|(1)-(2)\|}{(1)}$ cm)
	x	*y*	*x*	*y*	
1	488	30.5	484.88	23.73	0.109
2	549	30.5	542.84	23.49	0.074
3	610	30.5	608.29	24.17	0.035
4	671	30.5	671.32	24.42	0.025
5	732	30.5	736.11	24.35	0.024
6	793	30.5	796.44	23.47	0.021
7	457.5	91.5	456.75	87.34	0.044
8	518.5	91.5	516.22	87.75	0.034
9	579.5	91.5	575.82	86.18	0.036
10	640.5	91.5	637.75	87.95	0.019
11	701.5	91.5	703.29	86.42	0.019
12	762.5	91.5	766.8	85.91	0.020
13	823.5	91.5	824.58	86.1	0.014
14	457.5	152.5	454.96	150.7	0.020
15	518.5	152.5	514.43	152.5	0.023
16	579.5	152.5	574.83	149.27	0.027
17	640.5	152.5	640.64	149.46	0.012
18	701.5	152.5	703.11	150.98	0.007

	Actual Position		Image Position		
19	762.5	152.5	764.38	150.99	0.007
20	823.5	152.5	824.02	149.14	0.008
21	427	213.5	429.35	215.86	0.016
22	488	213.5	485.81	213.63	0.010
23	549	213.5	543.7	212.1	0.022
24	610	213.5	606.87	212.05	0.012
25	671	213.5	671.08	211.9	0.005
26	732	213.5	735.21	212.23	0.009



(a) (b)

Figure 7.8 The security patrolling task. (a) Images captured in the learning phase. (b) Images captured in the navigation phase.


(a)

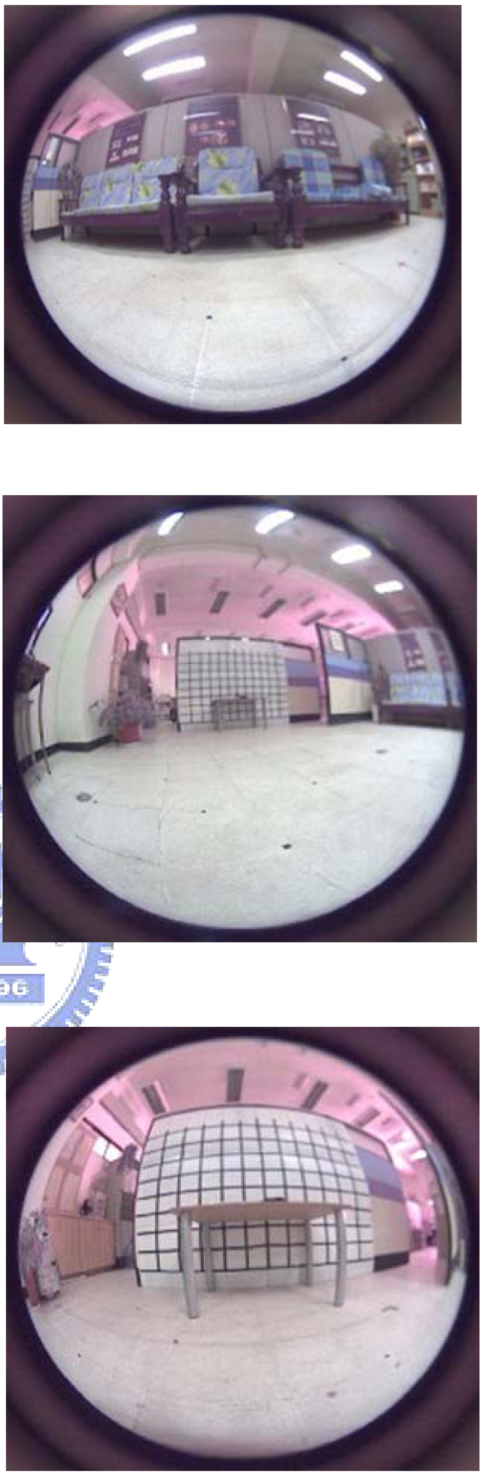

(b)

Figure 7.9 The security patrolling task. (a) Images captured in the learning phase. (b) Images captured in the navigation phase. (continued)

7.3 Discussions

The proposed system utilizes multiple vision-based autonomous vehicles to perform the security patrolling task. For this purpose, some monitoring points are utilized to guide the vehicles. By the way, there are more applications of the monitoring points, such as providing various services. Every monitoring point can be regarded, for example, as a business *service point* in which there are some customers. If the environment is a restaurant, the apparatus of showing menu can be equipped on the vehicles, and then the vehicles can move to each service point along assigned optimal paths to ask what dishes or services are needed. If the environment is a company, the vehicles also can be utilized to deliver documents or messages in each service point. Furthermore, if a walkable area can be divided into many ranges, in which each is within the controllable view of the vehicle, we may transform every range into a node such that the vehicles can arrive at anywhere in it to do some actions, such as detecting whether an unknown person has invaded with optimal randomized paths.

However, there are still some problems in the system. If an object appears next to the vehicle suddenly, the top-view omni-cameras will not have the ability to find out the vehicle. To solve the problem, it might be necessary to add information of color and sample models of the vehicles to this system. Furthermore, the vehicles are not on a plane, so the vehicle localization accuracy is affected by the heights of the vehicles. If the vehicle is taller and farther from the top-view omni-cameras, the error between the obtained centroid and the actual position of the vehicle will be large. However, we might be able to add an obvious mark on the center of the top of the vehicle. By finding the mark, the correct position can also be obtained. Finally, the proposed real-time collision avoidance technique between vehicles is feasible for two vehicles.

If the total number of vehicles is larger than two, we will need to consider the influence of passing points for the third vehicle. The problem is worth for future research.

Chapter 8
Conclusions and Suggestions for Future Works

8.1 Conclusions

In this study, we utilize multiple vision-based autonomous vehicles to develop a security patrolling system in an environment whose floor shape is composed of rectangular regions. We have proposed several techniques and adopt some algorithms which are summarized in the following.

(1) *An environment-information calculation method* has been proposed, by which we can obtain all rectangular regions, which form the floor shape of the patrolling environment, the turning points, and then all between-*MP* distances and paths. The turning points are utilized to enable the vehicles to move between any pair of *MP*s without collisions with the walls. With the turning points, we adopt *Dijkstra's algorithm* to obtain the shortest between-*MP* distances and paths between the two *MP*s which belong to the different regions.

(2) *A point-correspondence technique integrated with an image interpolation method for camera calibration* has been proposed. In this study, we don't use the traditional projection-based transformation. Instead, a grid pattern is used as the calibration target and corresponding points between 2-D image and 3-D global spaces are utilized. For the warped images captured by the top-view omni-cameras, the correct coordinate positions can be obtained by the

corresponding points and the use of an *image interpolation* method.

(3) *A faster point-correspondence technique* has been proposed. Because more corresponding points will yield better calibration accuracy, we adopt a *minimum mean square error (MMSE)* method to calculate quadratic curves and abundant cross points, in the image captured by the top-view omni-camera, can be obtained. Each cross point and its coordinates in the global space, obtained by an interpolation method, are exactly one pair of point correspondences.

(4) *A vehicle-pose learning method* has proposed, by which the vehicles are taught where and in which direction to perform the security monitoring task, which is to take pictures of monitored objects as defined in this study. Furthermore, the learned positions can be utilized to guide the vehicles.

(5) *An optimal method for randomized and load-balanced path planning* has been proposed, in which each *MP* is just passed once such that monitored objects can be patrolled uniformly. Additionally, the difference of the numbers of assigned *MP*s for all vehicles is smaller and a threshold distance is set to restrict the difference between path distances, so that the loads of all vehicles can be balanced. According to the numbers of assigned *MP*s, the *MP*s are chosen randomly, and then the system calculate the shortest paths with each *MP* on these paths appearing only once by the concept of the *TSP*.

(6) *A vehicle localization and monitoring method* has been proposed. Because the vehicles suffer from mechanic errors, we utilize the top-view omni-cameras to locate them in this study. By the odometer values of the vehicles, we can calculate the centroids of the vehicles in the image. After the centroids are transformed into the global space, the odometer values are corrected by the coordinates of the resulting points. Besides, the directional angles of the vehicles also must be corrected, in which two continuous correct position points are

utilized to do the job. Additionally, the cameras have the ability to monitor vehicles to see whether they are still under control. If any vehicle loses control of its action, the system will send an alarm message to the security center and stop all vehicles.

(7) *A real-time collision avoidance technique between two vehicles* has been proposed. By the odometer values, the system computes the distance between two vehicles in every cycle of a fixed-time duration and determines whether they are too close. If yes, the feasible alternative paths of the vehicles will be obtained by two different kinds of states, *path-intersecting* or *non-path-intersecting*.

The experimental results shown in the previous chapters have revealed the feasibility of the proposed system.

8.2 Suggestions for Future Works

The proposed strategies and methods, as mentioned previously, have been implemented on a vehicle system with multiple vision-based autonomous vehicles. According to this study, in the following we make several suggestions and point out some related interesting issues, which are worth further investigation in the future:

(1) using a pen-tilt-zoom camera equipped on the vehicle to capture clearer images, and then extracting features of the images to detect whether monitored objects still exist;

(2) adding the capability to detect more danger conditions;

(3) adding the capability of warning users immediately through cell phones or electronic mails;

(4) adding the capability of voice control when users want to issue navigation orders

to the vehicle;

(5) improving the real-time collision avoidance technique to be suitable for more vehicles; and

(6) improving the accuracy of finding the centroid of the vehicle.

References

[1] I. Fukui, "TV image processing to determine the position of a robot vehicle," *Pattern Recognition,* vol. 14, pp. 101-109, 1981.

[2] Betke M and Gurvits L, "Mobile robot localization using landmarks," *IEEE Transactions on Robotics and Automation*, vol. 13, no 2, pp 251-263,Apr., 1997.

[3] M. J. Magee and J. K. Aggarwal, "Determining the position of a robot using a single calibration object," *IEEE Conference on Robotics,* pp. 57-62, Atlanta, Georgia, USA, May 1983.

[4] J. Huang, C. Zhao, Y. Ohtake, H. Li, and Q. Zhao, "Robot position identification using specially designed landmarks," *Proceedings of 2006 IEEE Conference on Instrumentation and Measurement Technology*, Italy, Apr., 2006.

[5] H. L. Chou and W. H. Tsai "A new approach to robot location by house corners," *Pattern Recognition,* vol. 19, pp. 439-451, 1986.

[6] K. L. Chiang and W. H. Tsai, "Vision-based autonomous vehicle guidance in indoor environments using odometer and house corner location information," *Proceedings of 2006 IEEE International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pp. 415-418, USA, Dec. 18-20, 2006.

[7] K. C. Chen and W. H. Tsai, "A study on autonomous vehicle navigation by 2D object image matching and 3D computer vision analysis for indoor security patrolling applications," *Proceedings of 2007 Conference on Computer Vision, Graphics and Image Processing*, Miaoli, Taiwan, June, 2007.

[8] D. Cobzas, H. Zhang, and M. Jagersand, "Image-based localization with depth-enhanced image map," *Proceedings of IEEE International Conference on Robotics and Automation (ICRA 2003)*, pp. 1570-1575, Taipei, Taiwan, 2003.

[9] P. Biber, H. Andreasson, T. Duckett, and A. Schilling, "3D modeling of indoor

environments by a mobile robot with a laser scanner and panoramic camera," *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2004)*, Sendai, Japan, Sept. 28 - Oct. 2, 2004.

[10] C. C. Lai and W. H. Tsai, "A study on automatic indoor navigation techniques for vision-based mini-vehicle with off-line environment learning capability," *Proceedings of 2003 Conference on Computer Vision, Graphics and Image Processing*, Kinmen, Taiwan, June, 2003.

[11] N. Winters and J. Santos-Victor, "Mobile robot navigation using omni-directional vision," *Proceedings of. 3rd Irish Machine Vision and Image Processing Conference (IMVlP'99)*, Dublin, Ireland, 1999.

[12] L. E. Kavraki, J. C. Latombe, P. vestka, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, Aug., 1996.

[13] R. Kimmel, N. Kiryati, and A. M. Bruckstein, "Multivalued distance maps for motion planning on surfaces with moving obstacles," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 3, pp. 427-436, June, 1998.

[14] Y. Mezouar and F. Chaumette, "Path planning for robust image-based control," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 4, Aug., 2002.

[15] S. Hert and B. Richards, "Multiple-Robot Motion Planning = Parallel Processing + Geometry," *Springer-Verlag*, pp. 195-215, London, UK, 2000.

[16] M. C. Chen and W. H. Tsai "Vision-based security patrolling in indoor environments using autonomous vehicles," *Proceedings of 2005 Conference on Computer Vision, Graphics and Image Processing,* Taipei, Taiwan, Aug., 2005.

[17] D. Parsons and J. Canny, "A motion planner for multiple mobile robots," *IEEE International Conference on Robotics and Automation,* vol. 1, pp. 8-13, May 13-18, 1990.

[18] M. Bennewitz and W. Burgard, "Coordinating the motions of multiple mobile robots using a probabilistic Model," *Proceedings of the International Symposium on Intelligent Robotic Systems (SIRS)*, England, 2000.

[19] F. M. Marchese and M. D. Negro, " Path-Planning for Multiple Generic-shaped Mobile Robots with MCA," *Springer-Verlag Ed.*, vol. 3993, pp 264-271, Berlin, Heidelberg(D), May, 2006.

[20] N.L.J. Ulder, E. H. L. Aarts, H. J. Bandelt, P. J. M. Van Laarhoven, and E. Pesch, "Genetic local search algorithms for the traveling salesman problem," *Springer*, *Parallel Problem Solving from Nature—Proceedings of 1st Workshop* , vol. 496, pp. 109-116, Berlin, Germany, 1991.

[21] S. Tschoke,R. Lubling, and B .Monien, "Solving the Traveling Salesman Problem with a Distributed Branch-and-Bound Algorithm on a 1024 Processor Network," *Proceedings of International Parallel Processing Symposium*, April 25-28, 1995, pp. 182-189.

[22] J. W. Pepper, B. L. Golden, and E. A. Wasil, "Solving the traveling salesman problem with annealing-based heuristics: A computational study," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 32, no. 1, Jan., 2002.

[23] D. A. Forsyth and J. Ponce, *Computer Vision: A Modern Approach*, Prentice Hall, Upper Saddle River, New Jersey, USA, 2002.

[24] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd edition. MIT Press, Cambridge, 2001.