

國立交通大學

多媒體工程研究所
碩士論文

利用環場攝影機於車內做基於擴增實境技術的室外園
區導覽

In-car Tour Guidance in Outdoor Park Areas Based on
Augmented Reality Techniques Using an Omni-camera

研究生：衛彥成

指導教授：蔡文祥 教授

中華民國 一〇二年六月

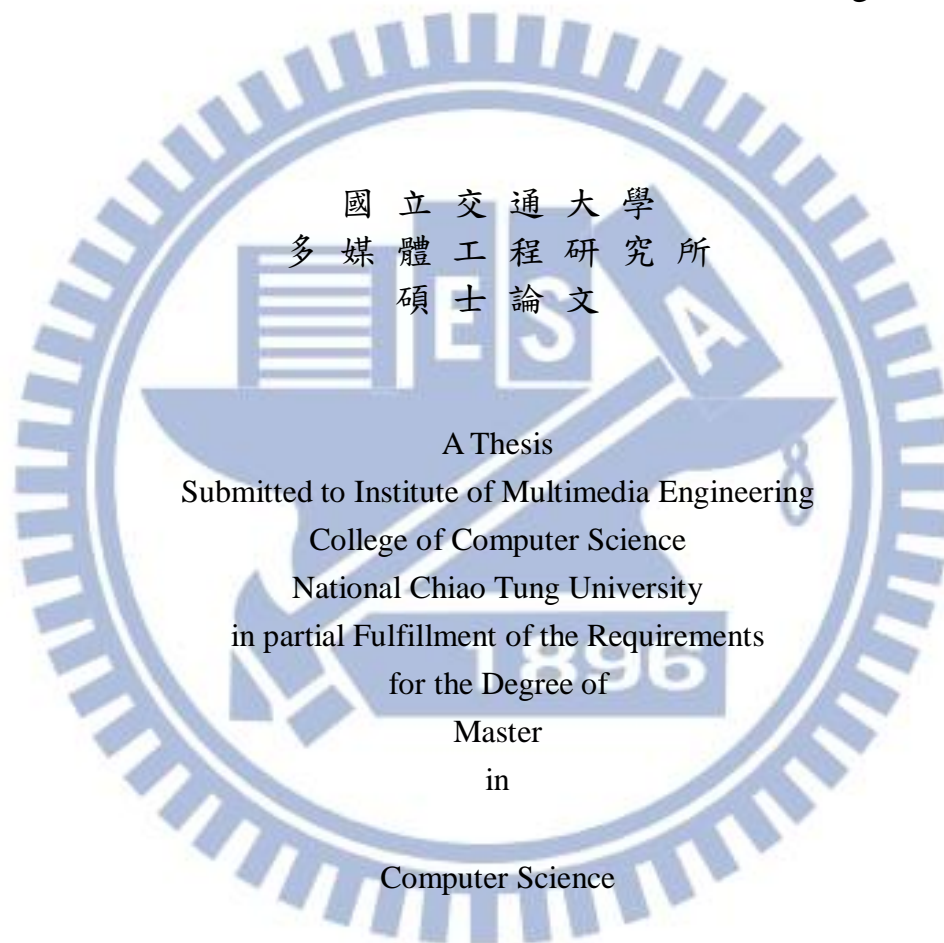
利用環場攝影機於車內做基於擴增實境技術的室外園區導覽
In-car Tour Guidance in Outdoor Park Areas Based on Augmented
Reality Techniques Using an Omni-camera

研 究 生：衛彥成

Student : Yen-Cheng Wei

指 導 教 授：蔡文祥

Advisor : Wen-Hsiang Tsai



June 2013

Hsinchu, Taiwan, Republic of China

中華民國一〇二年六月

In-car Tour Guidance in Outdoor Park Areas Based on Augmented Reality Techniques Using an Omni-camera

Student: Yen-Cheng Wei

Advisor: Wen-Hsiang Tsai

Institute of Multimedia Engineering,

College of Computer Science

National Chiao Tung University

ABSTRACT

In this study, an augmented-reality based tour guidance system for use in park areas using a vehicle and computer vision techniques has been proposed. With the proposed system, a user in the vehicle driven in a park area can get from the system tour guidance information about the names of nearby buildings along the path. The building names are augmented on the passenger-view image which is displayed on the mobile device held by the user in the vehicle.

To implement the proposed system with the augmented reality function, firstly an environment map is generated in the learning phase, which includes the information about the path of the tour, the along-path line features which can be detected for vehicle localization, and the building names. All the data are learned either manually or semi-automatic, and saved into the database for use in the navigation phase.

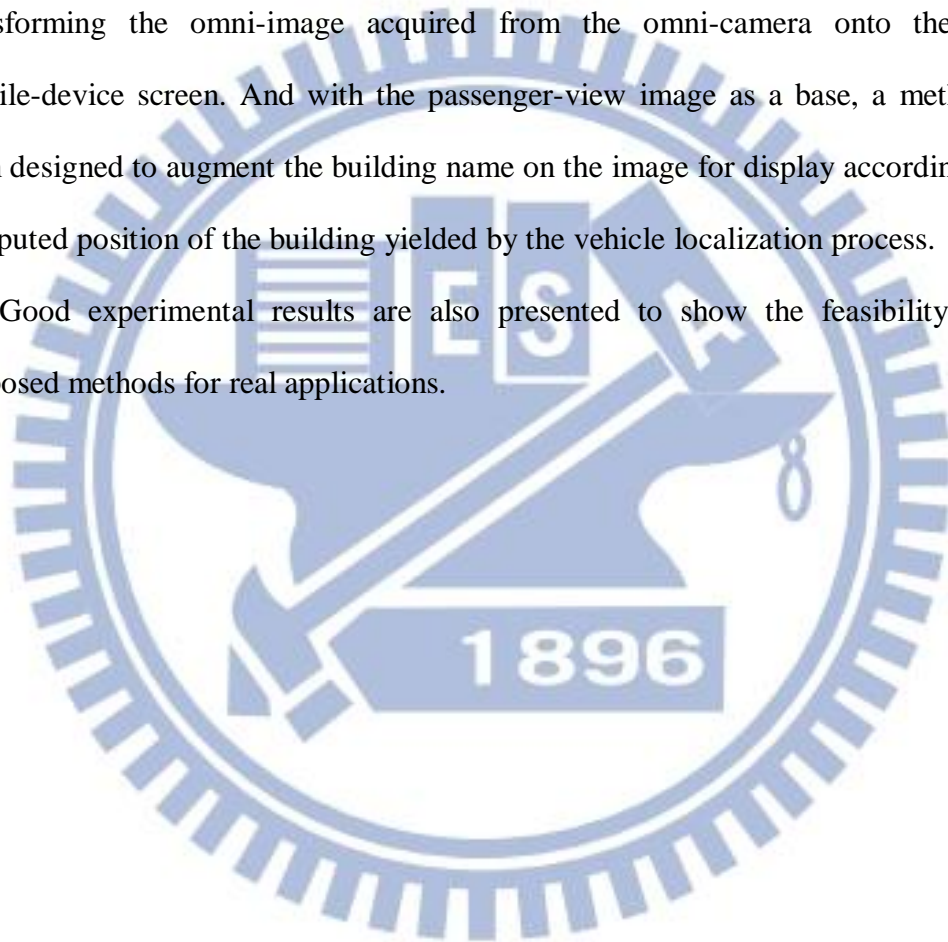
Secondly, a method for detecting along-path vertical-line features which appear in the omni-image is used to localize the vehicle. The method detects edges in the input omni-image, and analyzes them to detect continuous or broken vertical lines

with widths.

Next, a method for vehicle localization is proposed, which analyzes the detected line features and computes the vehicle position by a modified longest common subsequence algorithm. Meanwhile, motion vectors are used to estimate the vehicle speed for locating the vehicle when there is no detectable feature around the vehicle.

Finally, a method is proposed for generating a passenger-view image by transforming the omni-image acquired from the omni-camera onto the user's mobile-device screen. And with the passenger-view image as a base, a method has been designed to augment the building name on the image for display according to the computed position of the building yielded by the vehicle localization process.

Good experimental results are also presented to show the feasibility of the proposed methods for real applications.



利用環場攝影機於車內做基於擴增實境技術的室外園 區導覽

研究生：衛彥成

指導教授：蔡文祥 博士

國立交通大學多媒體工程研究所

摘要

本研究利用車輛與電腦視覺技術，建立一個基於擴增實境(augmented reality; AR)技術的室外園區導覽系統。利用此系統，使用者開車在園區時可以接收到導覽資訊，例如路徑上車輛周圍之建築物名稱。此導覽資訊出現在車內使用者的手持裝置上影像的建築物上。

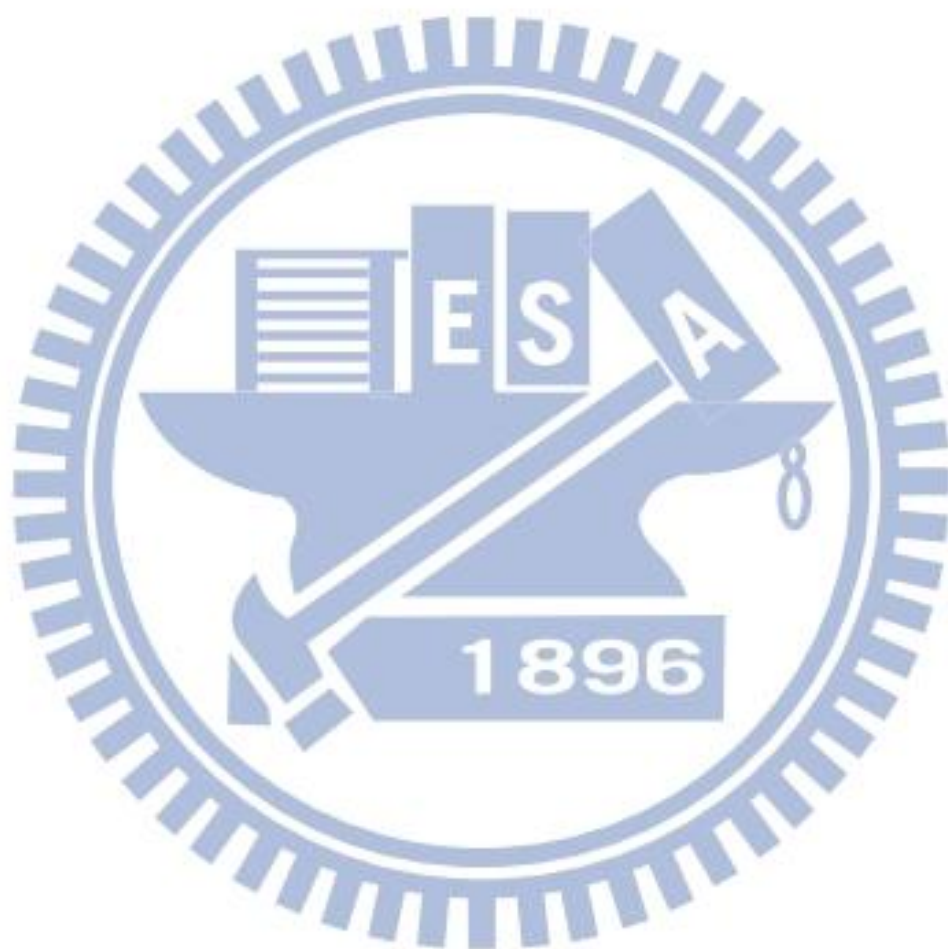
為實現此一擴增實境導覽系統，本研究首先在學習階段建立環境地圖，地圖中包含導覽的路徑、周圍欲偵測之垂直線特徵、附近建築物所在位置及其名稱等資料。所有資料經過手動或半自動的學習之後，會儲存至一資料庫以供導航階段使用。

接下來，本研究提出一個利用環場影像偵測路徑上車輛周圍的垂直特徵物進行車輛定位的方法，該系統利用環場攝影機取得之影像來分析垂直線，垂直線即使不連續或是有寬度仍然能夠偵測出來。

此外，本研究利用最長共同子序列的演算法分析偵測到的直線，並計算出車輛的位置，來對車輛進行定位。若無法用特徵物做定位，則利用動態向量(motion vector)來預估車輛速度，藉以計算車輛所到達位置。

最後，本研究利用環場攝影機所取得的影像經過轉換後，產生出模擬副駕駛看出車外的影像，並且將此影像顯示在手持裝置上。同時，以此影像為基底，並利用車輛定位的結果，將建築物在影像上的位置計算出來，藉以疊加此建築物的名稱在該影像上。

上述方法經過實驗得到良好的結果，顯示本研究所提出的系統確實可行。

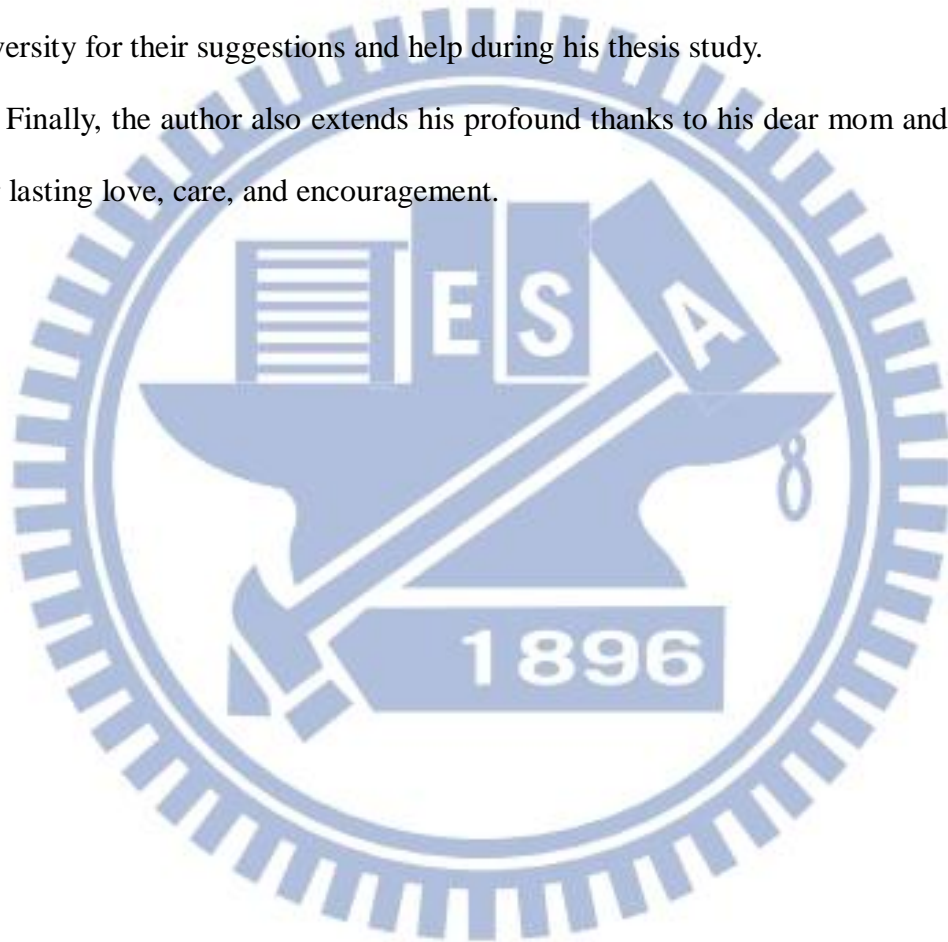


ACKNOWLEDGEMENTS

The author is in hearty appreciation of the continuous guidance, discussions, and support from his advisor, Dr. Wen-Hsiang Tsai, not only in the development of this thesis, but also in every aspect of his personal growth.

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 his thesis study.

Finally, the author also extends his profound thanks to his dear mom and dad for their lasting love, care, and encouragement.



CONTENTS

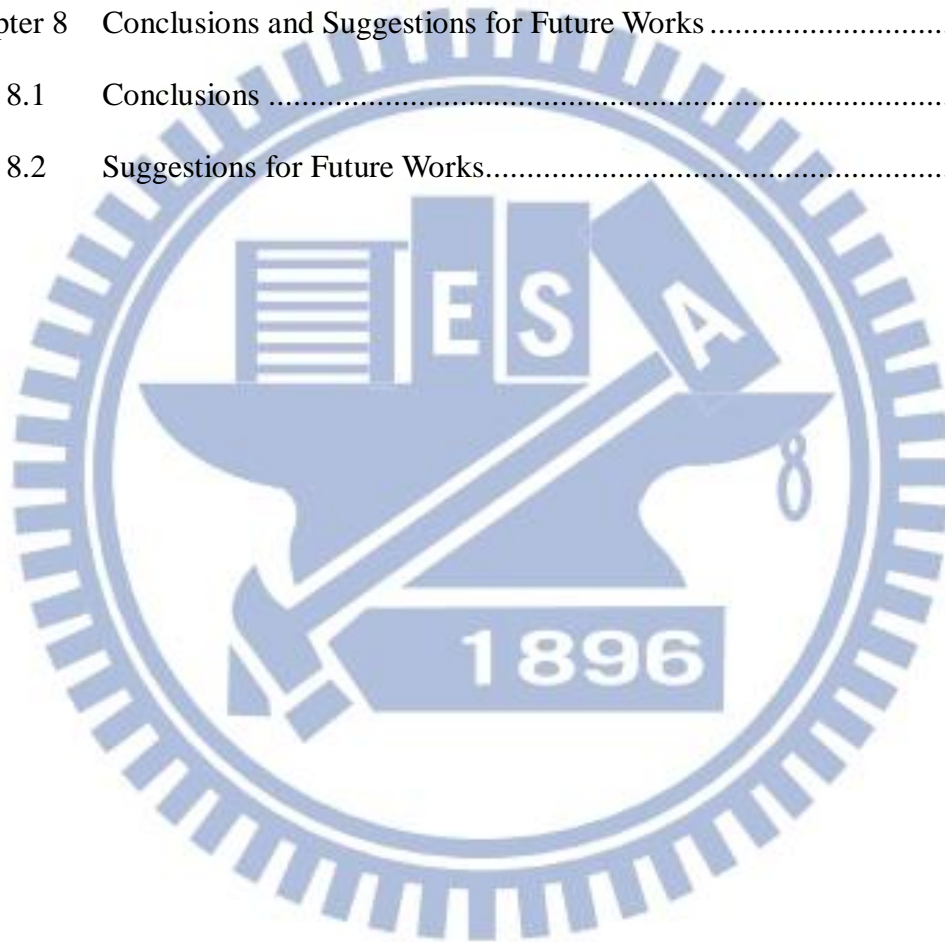
ABSTRACT (in English)	i
ABSTRACT (in Chinese)	iii
ACKNOWLEDGEMENTS	v
CONTENTS	vi
LIST OF FIGURES	x
LIST OF TABLES	xvi

Chapter 1	Introduction	1
1.1	Background and Motivation	1
1.2	Survey of Related Studies	4
1.3	Overview of Proposed Methods	5
1.3.1	Terminologies	5
1.3.2	Brief Descriptions of Proposed System	5
1.4	Contributions	8
1.5	Thesis Organization	9
Chapter 2	System Design and Processes	10
2.1	Ideas of Proposed Method	10
2.2	System of Configuration	12
2.2.1	Hardware Configuration	13
2.2.2	Software Configuration.....	15
2.2.3	Network Configuration.....	16
2.3	Network System.....	17
2.3.1	Server-side System	17
2.3.2	Client-side System.....	18
2.3.3	Cooperation between Client and Server Sides	18
2.4	System Processes	20
2.4.1	Learning Process	20

2.4.2	Navigation Process	21
Chapter 3	Learning of Environments	23
3.1	Ideas of Proposed Environment Learning Techniques.....	23
3.2	Coordinate Systems Used in This Study	24
3.3	Construction of Environment Map	25
3.3.1	Information Included in Environment Map	26
3.3.2	Creation of Database for Environment Map	27
3.4	Learning of Environment Features	28
3.4.1	Learning of Navigation Paths.....	28
3.4.2	Learning of Vertical Lines in Environments	30
3.4.3	Learning of Building Information	34
3.5	Experimental Results	35
Chapter 4	Automatic Detection of Vertical Lines in Environments with an Omni-camera.....	37
4.1	Introduction	37
4.2	Idea of Analysis of Vertical Lines in Omni-images	38
4.3	Detection of Vertical Lines in Environments.....	44
4.3.1	Initial Detection by Canny Edge Detector.....	44
4.3.2	Detection of Lines with Widths.....	45
4.3.3	Detection of Broken Lines	47
4.4	Algorithm for Vertical Line Detectin	49
4.5	Experimental Results	51
Chapter 5	Vehicle Localization for Tour Guidance in Outdoor Park Areas by Computer Vision Techniques	54
5.1	Introduction	54
5.2	Estimation of Vehicle Speed.....	55

5.2.1	Computation of Motion Vectors.....	55
5.2.2	Vehicle Speed Estimation Using Motion Vectors.....	57
5.3	Vehicle Localization by Single Line Features	58
5.3.1	Idea of Vehicle Localization by Line Features.....	59
5.3.2	Algorithm for Vehicle Localization by Single Line Features	60
5.4	Vehicle Localization by Multiple Line Features.....	61
5.4.1	Review of Longest Common Subsequence (LCS) Algorithm....	61
5.4.2	Vehicle Localization using Multiple Features by LCS Algorithm	63
5.5	Knowledge-based Analysis of Tours.....	64
5.5.1	Uses of Knowledge about Environments.....	64
5.5.2	Algorithm for Vehicle Localization in Tours	66
5.6	Experimental Results	67
Chapter 6	Proposed Augmented Reality-Based Tour Guidance Using an	
	Omni-camera.....	70
6.1	Ideas of Proposed Techniques.....	70
6.2	Construction of Images from Front Passenger's View.....	71
6.2.1	Construction of Image-to-space Mapping Table	71
6.2.2	Review of Adopted Method for Perspective-view Image	
	Generation	79
6.2.3	Review of Generation of Perspective-mapping Table	82
6.2.4	Generation of Passenger-view Image	84
6.3	Augmenting Names of Buildings on Passenger-view Images.....	88
6.3.1	Calculating Positions of Buildings in Passenger-view Images ...	89
6.3.2	Algorithm of Augmenting Names of Buildings on Images	90
6.4	Tour Guidance in Park Areas	93

6.4.1	Ideas of Tour Guidance in Park Areas	93
6.4.2	Algorithm for Tour Guidance in Park Areas	93
6.5	Experimental Results	95
Chapter 7	Experimental Results and Discussions.....	97
7.1	Experimental Results	97
7.2	Discussions	101
Chapter 8	Conclusions and Suggestions for Future Works	103
8.1	Conclusions	103
8.2	Suggestions for Future Works.....	104



LIST OF FIGURES

Figure 1.1 Proposed AR-based tour guidance system. (a) Image of the used vehicle. (b) Illustration of proposed AR-based guidance system working on a moving vehicle.....	3
Figure 1.2 A flowchart of proposed learning process.	7
Figure 1.3 A flowchart of proposed navigation process.....	8
Figure 2.1 The video surveillance vehicle used in this study with an omni-camera affixed on the car roof. (a) A front view of the vehicle. (b) A side view of the vehicle.....	10
Figure 2.2 Positions of the omni-imaging device affixed to the video surveillance vehicle roof and the corresponding FOV. (a) The device is affixed at the rear-middle of the car roof. (b) The device is affixed at the right-front of the car roof.....	11
Figure 2.3 Structure of the proposed surveillance system.	13
Figure 2.4 The component of the camera device and entire device. (a) AISYS ALTAIR U500C cameras. (b) JHF8M-5MP lens. (c) Entire camera device.	15
Figure 2.5 The architecture of the local network used in this study.	16
Figure 2.6 Cooperation between client and server sides.....	19
Figure 2.7 Flowchart of learning process.....	21
Figure 2.8 Flowchart of proposed tour guidance system.	22
Figure 3.1 The three coordinate systems used in the proposed system. (a) The global coordinate system. (b) The camera coordinate system. (c) The image coordinate system.....	25

Figure 3.2 User interface for real-world map construction by use of OpenStreetMap.	26
Figure 3.3 The real-world map we use in the proposed system.	27
Figure 3.4 The vehicle on a path while detecting a feature. (a) An illustration of the vehicle driving on the path. (b) An omni-image with a detected feature — a light pole.	30
Figure 3.5 Illustrations of multiple feature detection. (a) Illustration of detected feature on the map. (b) The angle of features detected. (c) Illustration of calculating the angle of θ_f	32
Figure 3.6 Learning of buildings. (a) An illustration of learning the building. (b)The result of learning the building in the map.	34
Figure 3.7 The environment map we use in the proposed system.	36
Figure 3.8 The environment map with the path.	36
Figure 3.9 The environment map with the path and features.	36
Figure 4.1 Camera and image coordinate systems.	40
Figure 4.2 Illustration of a space line projected on to the image plane.	41
Figure 4.3 An illustration of detection of lines with widths in the image. The black boxes are line points; the yellow area is the region we define as a line; and the red line specifies the direction of this line.	46
Figure 4.4 An illustration of a line detection. The black painted box is line points. The red painted box is the scan point P_s . The green painted box is the neighbor points P_n	47
Figure 4.5 An illustration of a broken line detection. (a) A broken line with a 80% density. (b) Two lines whose overall density is 80%.	48
Figure 4.6 An illustration of vertical lines in an omni-image. The area of red points are the center of the omni-image, and each green line corresponds to a	

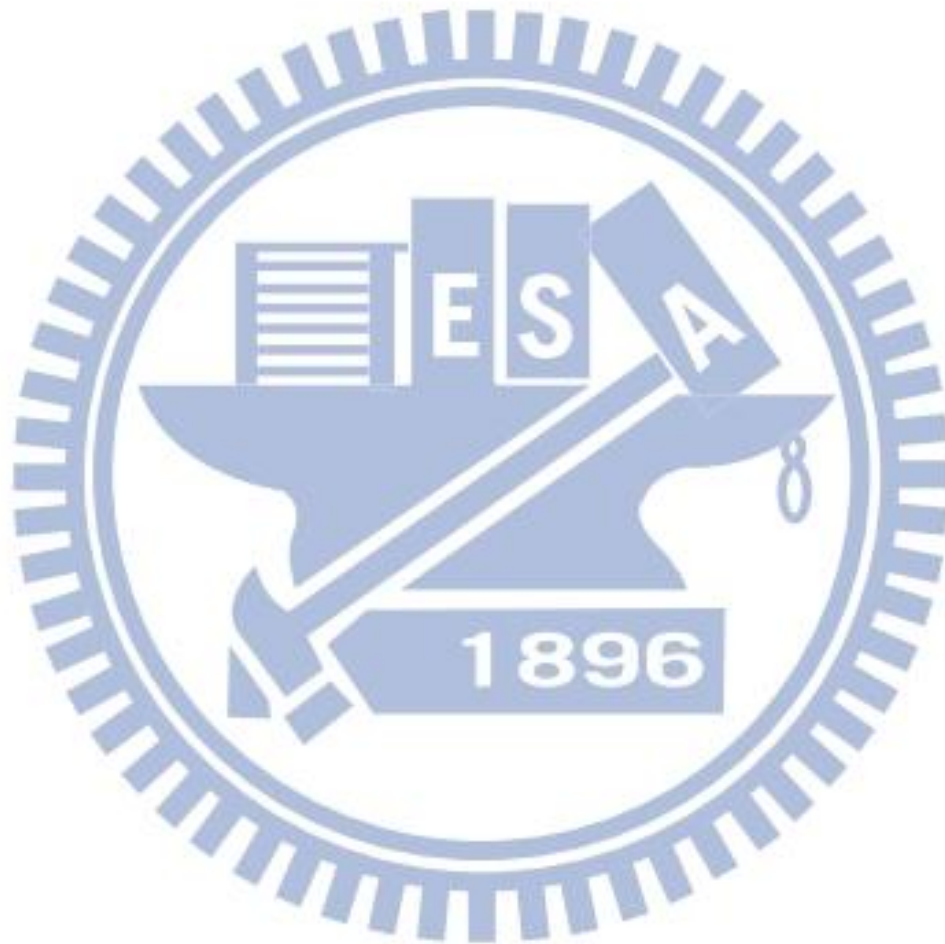
vertical line in the real-world space.	50
Figure 4.7 Illustrations of the phenomenon that a vertical line become a radial line in the omni-image. (a) Scene 1. (b) Scene 2.....	52
Figure 4.8 Results of Canny edge detection. (A) Result of Figure 4.7(a). (B) Result of Figure 4.7(b).	52
Figure 4.9 Results of vertical line detection. (a) Result obtained from Figure 4.8(a). (b) Result obtained from Figure 4.8(b).	53
Figure 5.1 An illustration of searching for the best-match macroblock.....	55
Figure 5.2 An illustration of cutting a part of the omni-image for motion vector computation. The area enclosed by the red line is the part we cut.	57
Figure 5.3 An illustration of locating the vehicle. (a) An omni-image with a detected line feature. (b) An illustration of locating the vehicle on the map.	59
Figure 5.4 An Illustration Of Omni-Image. (A) The Omni-Image Which Divided Two Parts. (B) Two parts of image and the purple line is direction of system to detect the line features.	65
Figure 5.5 Detected motion vectors in an omni-image. (a) The original image. (b) The detected motion vectors.	68
Figure 5.6 The localization of vehicle by using single feature. (a) The omni-image acquired from the camera. (b) A binary omni-image, in which the red line is the detected line feature. (c) The map showing the position of vehicle where the red point is the position of the vehicle and the blue points are the positions of features.	68
Figure 5.7 Vehicle localization by using multiple features. (a) The omni-image acquired from the camera. (b) A binary omni-image, in which the red lines indicate detected line features. (c) The map showing the position of vehicle, where the red point is the position of the vehicle and the blue	

points are the positions of the detected features.	69
Figure 6.1 The space points and their corresponding image points.....	72
Figure 6.2 Finding out the focal point O_m	73
Figure 6.3 The interface for acquiring the data of the world space points.....	74
Figure 6.4 Nonlinear property of an omni-camera with mirror surface shape.....	75
Figure 6.5 Mapping between pano-mapping table and omni-image.....	77
Figure 6.6 Creation of pano-mapping table.....	78
Figure 6.7 A Top-view configuration for generating a perspective-view image.....	79
Figure 6.8 A lateral-view configuration for generating a perspective-view image. ..	82
Figure 6.9 Illustration of construction of a perspective-mapping table. (a) A Top-view configuration for generating a perspective-mapping table. (b) A lateral-view configuration for generating a perspective-view image.....	84
Figure 6.10 An illustration of viewpoint in the vehicle. (a) Top-view of the vehicle where the blue star is the viewpoint and the red line is the region of the view. (b) Side-view of the vehicle where again the blue star is the viewpoint and the red line is the region of view.....	85
Figure 6.11 An illustration of viewpoints through the windshield. (a) The left side angles, where the yellow line is a horizontal line with an angle of zero, and the red line is the boundary of the viewpoint. (b) The right side angles. (c) The upside angles, where the yellow line is a vertical line with an angle of zero, and the red line is the boundary of the viewpoint. (d) The downside angles. (e) all view of the viewpoint.	86
Figure 6.12 An illustration of shifting the viewpoint. (A) Top-view of shifting the viewpoint where the blue star is the viewpoint we set and the green star is the viewpoint of camera. (B) Side-view of shifting the viewpoint where the blue star is the viewpoint we set and the green star is the viewpoint of	

camera.....	87
Figure 6.13 Illustration of construction of passenger-view images. (a) A Top-view configuration for generating a passenger-view image. (b) A lateral-view configuration for generating a passenger-view image.	89
Figure 6.14 An illustration for calculating the angle of the direction to the building.	90
Figure 6.15 An illustration for calculating the position of the building.....	90
Figure 6.16 An illustration of calculating the building position.....	91
Figure 6.17 An illustration of the view of the image. (a) The entire building can be seen in the image. (b) Only part of the building can be seen in the image.	91
Figure 6.18 (a)The omni-image acquired from the omni-camera. (b)The passenger-view image transformed from (a). (c)The omni-image acquired from the omni-camera. (d)The passenger-view image transformed from (c)	95
Figure 6.19 Two passenger-view images with the building names augmented.	96
Figure 7.1 The environment map we use in the proposed system.....	97
Figure 7.2 An experimental result of the learning stage. (a) An image of the vehicle driven on the path and detecting the line feature. (b) An omni-image acquired from the omni-camera. (c) A line feature detected by the system.	98
Figure 7.3 An experimental result of detecting the line features and locating the vehicle. (a) An omni-image acquired from the omni-camera. (b) Another omni-image acquired from the omni-camera. (c) A line feature detected by the system. (d) Another line feature detected by the system. (e) The location of the vehicle computed by the system and indicated by the red point. (f) Another location of the vehicle computed by the system and	

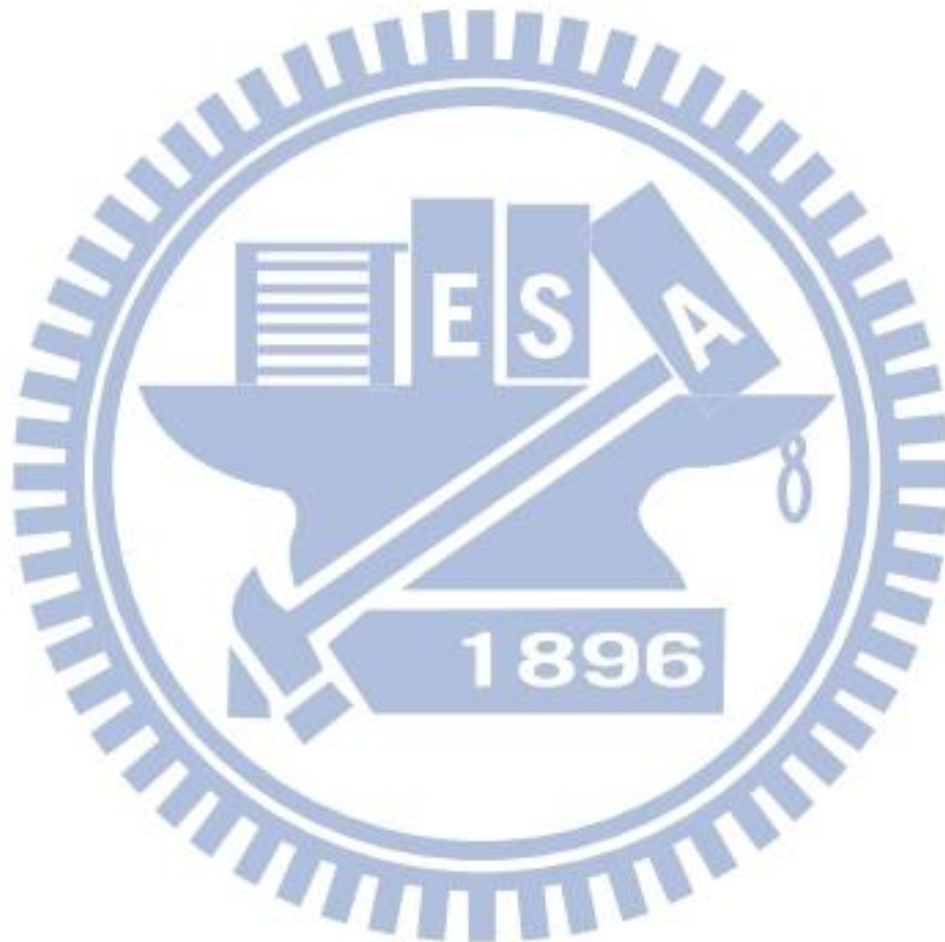
indicated by the red point. 99

Figure 7.4 AR-based navigation. (a) An image of the vehicle on the path. (b) An omni-image acquired with the omni-camera. (c) The passenger-view image with an augmented building name. (d) Another image of the vehicle on the path. (e) Another omni-image acquired with the omni-camera. (f) Another passenger-view image augmented with the building name.100



LIST OF TABLES

Table 2.1 Specifications of the laptop computers and the pad used in this study.....	14
Table 2.2 Specification of the CMOS cameras used in the imaging device	15
Table 2.3 Specification of the lens used in the imaging device.....	15
Table 6.1 Example of pano-mapping table of size $M \times N$	73



Chapter 1

Introduction

1.1 Background and Motivation

With the advance of technology, video cameras are widely used in many applications that bring convenience in our daily life. For instance, a vehicle equipped with on-top video cameras can help a driver to monitor surrounding environments and to be aware of dangerous situations so that car accidents could be avoided. Furthermore, if people drive cars which are equipped with video cameras working like digital event recorders, then when car accidents occur, they can clarify the responsibility by checking the recorded video.

Most researches of vision-based techniques are based on the use of traditional projective video cameras, but the limited field of view (FOV) of this type of camera causes some problems. For example, if we want to see all the views surrounding the car, we need more than four projective cameras in general. This needs more cost and superfluous computation time. Thus, we choose to use an omni-camera to be our imaging equipment in this study.

Moreover, we can use this camera system to develop interesting and useful applications by combining real-world images captured from the cameras and augmenting them with guidance information created by computers for the purpose of tour guidance in outdoor environments. In other words, the real-world environment can be augmented by computer-generated information (labels, texts, objects, etc.) to enhance the perception of the real world, and this is the so-called *augmented reality (AR)-based tour guidance*.

In more detail, the AR technique can help implementing a car navigation system which provides the driver with information of roads and surrounding buildings by projecting the names on the windshield or on a display device such as an iPad [1] in an AR way. There exist products of head-up displays (HUDs) on the windshield, which can show information like the vehicle speed and the engine speed. With the HUD, the driver can easily catch the information from the HUD device instead of looking down at the dashboard, allowing he/she to focus on driving without being disturbed. Sometimes, the latter action of looking down is the reason of a car accident. In addition, although AR techniques based on the Global Positioning System (GPS) are getting popular nowadays, sometimes they are difficult to utilize for the purpose of car positioning because of their imprecision in positioning with errors ranging from 3 to 15 meters. Also, the GPS does not work in tunnels or inside buildings.

Therefore, we propose in this study to integrate the uses of omni-cameras and AR techniques with a vehicle to implement a more accurate and effective non-GPS guidance system for driving tours in park areas. Furthermore, instead of using the HUD device for displaying the augmented image, Chen and Tsai [1] showed the augmented image on an iPad and projected the image onto the car windshield for the driver to inspect during driving without looking down. In this study, we, however, assume that the AR-based guidance information is to be inspected by a passenger sitting in the car, so we display the guidance information on the screen of an iPad held by the passenger all the time during the driving guidance session. Moreover, in order to get the information of an environment map, we use computer vision techniques to obtain the positions of nearby buildings via analysis of the features in the omni-images acquired by the omni-camera.

In summary, the research goal in this study is to develop a tour guidance system for use by passengers in cars. To accomplish this goal, we use a vehicle equipped

with an omni-camera on the vehicle top as an experimental platform. Also, we use an iPad as a display device for showing the augmented image. The iPad is held by a passenger sitting in the car.

An image of the used vehicle is shown in Fig. 1.1(a) and an illustration of the proposed system is shown in Fig. 1.1(b). Listed below are more detailed descriptions of the desired capabilities of the proposed AR-based park-area guidance system.



Figure 1.1 Proposed AR-based tour guidance system. (a) Image of the used vehicle. (b) Illustration of proposed AR-based guidance system working on a moving vehicle.

1. The system can learn the environment map automatically.
2. The system can detect vertical line features in the environment automatically and measure their positions. It then marks the relative locations of the vehicle with respect to the line features on the environment map.
3. The system is capable of computing the accurate vehicle position so that relevant augmented information can be computed accordingly and displayed on the iPad at correct locations.
4. The image appearing on the iPad shows in an AR way as a combination of the real-world image and the nearby building names for tour guidance.

1.2 Survey of Related Studies

In this section, we give a survey of related studies, such as video surveillance, design of omni-cameras for uses on vehicles, vehicle navigation, AR techniques, etc.

In recent years, video surveillance for various applications has been studied more intensively. For instance, Trivedi et al. [2] proposed methods to enhance driving safety by video surveillance systems using omni-cameras. In addition, Jeng and Tsai [3, 4] proposed a method based on the concept of pano-mapping table to calibrate omni-cameras without knowing the extrinsic parameters of the omni-cameras. Moreover, a new type of omni-vision system designed by combining two projective cameras and two mirrors attached back to back was proposed in Kuo and Tsai [5].

Furthermore, a lot of methods for vehicle navigation by using landmarks have been proposed. Betke and Gurvits [6] proposed a localization method to identify surrounding landmarks and find their locations in an environment map. For detection of landmarks in omni-images, Ho and Chen [7] proposed an algorithm to detect ellipses, and Wu and Tsai [8] proposed a method which uses the features of lines to localize the vehicle.

In a similar work, Grosch [9] proposed a method for vehicle navigation in the indoor environment by using panoramic images. Moreover, more and more outdoor environment applications using AR techniques have been proposed. Lee et al. [10] proposed a method using omni-camera to conduct object tracking in outdoor environments, simulating the user's view with AR techniques. Reitmayr and Drummond [11] proposed a model-based tracking system for augmented reality in urban environments by using handheld devices. Furthermore, Sandor et al. [12] proposed a method that uses AR techniques for delivering information to a driver by a head-up display device.

1.3 Overview of Proposed Methods

1.3.1 Terminologies

The definitions of some related terms used in this study are described as follows.

1. Omni-camera: an omni-camera has a mirror with a hyperboloidal or other geometric shape in front of a conventional projective camera, which projects a 360-degree surrounding scene onto the camera's imaging plane to form an omni-image by the mirror surface reflection function.
2. Omni-image: the image captured with an omni-camera device.
3. Video surveillance vehicle: a car with an omni-camera equipped on the top of the car.
4. Environment map: a real-world map constructed by "OpenStreetMap" (an open source for the Internet) and including features learned by our system.
5. Features of vertical lines: features in the real world which has vertical-line shapes such as light poles on street sidewalks, edge lines on building walls, etc.
6. Perspective-view image: an image originally acquired with the omni-camera and later perspectively-transformed into another image as it is seen by the human eye.

1.3.2 Brief Descriptions of Proposed System

There are four goals in the proposed system as described in the following.

1. The system is able to learn the environment objects automatically such as defined features, navigation paths, and building information on the map.
2. The system is able to detect vertical lines in outdoor environments.
3. The proposed system is able to compute the information of the position of the

vehicle.

4. The system is able to generate the passenger-view image for viewing on an iPad and augment nearby building names on it.

In order to achieve the above goals, the system operations can be divided into two phases: the learning process and the navigation process. The following are the major steps of the learning process.

1. Construct a real-world map by the open source “OpenStreetMap” available on the Internet.
2. Select a path on the map, and specify manually as nodes the positions of landmarks (such as street light poles or wall edges on buildings), whose features are usable for vehicle localization, along the path on the map.
3. Set up the omni-camera on the top of video surveillance vehicle on the front-right corner, drive the vehicle to follow every node along the path, acquire an image of the node environment, “learn” (1) the features of the corresponding landmark in the acquired image and (2) the buildings which are located around the vehicle, and record the data into a database.
4. Calculate the corresponding relation of the path and the features automatically.

A brief illustration of the above learning process is shown in Fig. 1.2. And the following are the major steps of the proposed navigation process.

1. Set up the omni-camera on the top of video surveillance vehicle on the front-right corner.
2. Load environment map information learned in advance and the mapping table for the omni-image into the system.
3. Detect vertical line features in the surround of the video surveillance vehicle using the omni-camera.
4. Use the detected features to calculate the position of the vehicle on the

environment map.

5. Calculate the position of the nearby building on the map by using the computed location of the vehicle.
6. Generate the passenger-view by transforming the omni-images acquired by the omni-camera.
7. Augment the building name on the passenger-view image.
8. Repeat the above steps until the vehicle reaches a pre-selected destination.

A brief illustration of the above navigation process is shown in Fig. 1.3.

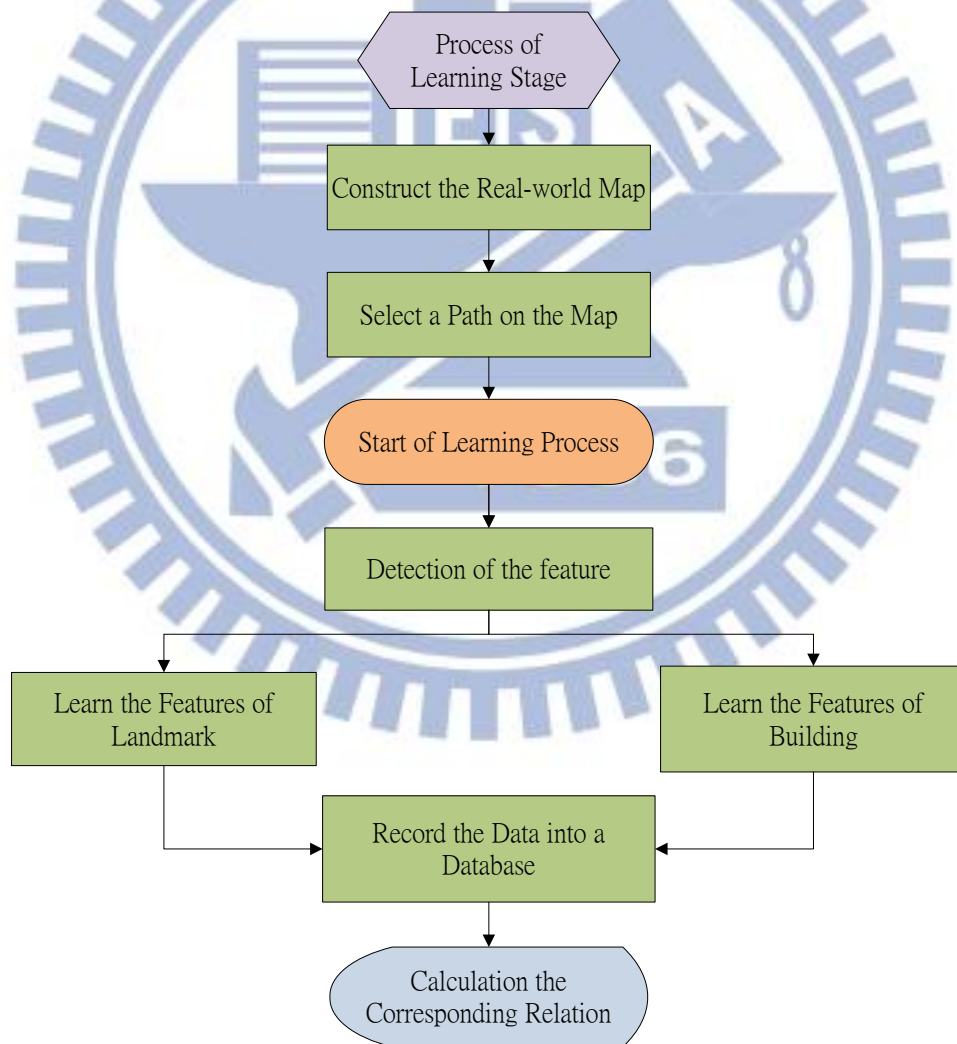


Figure 1.2 A flowchart of proposed learning process.

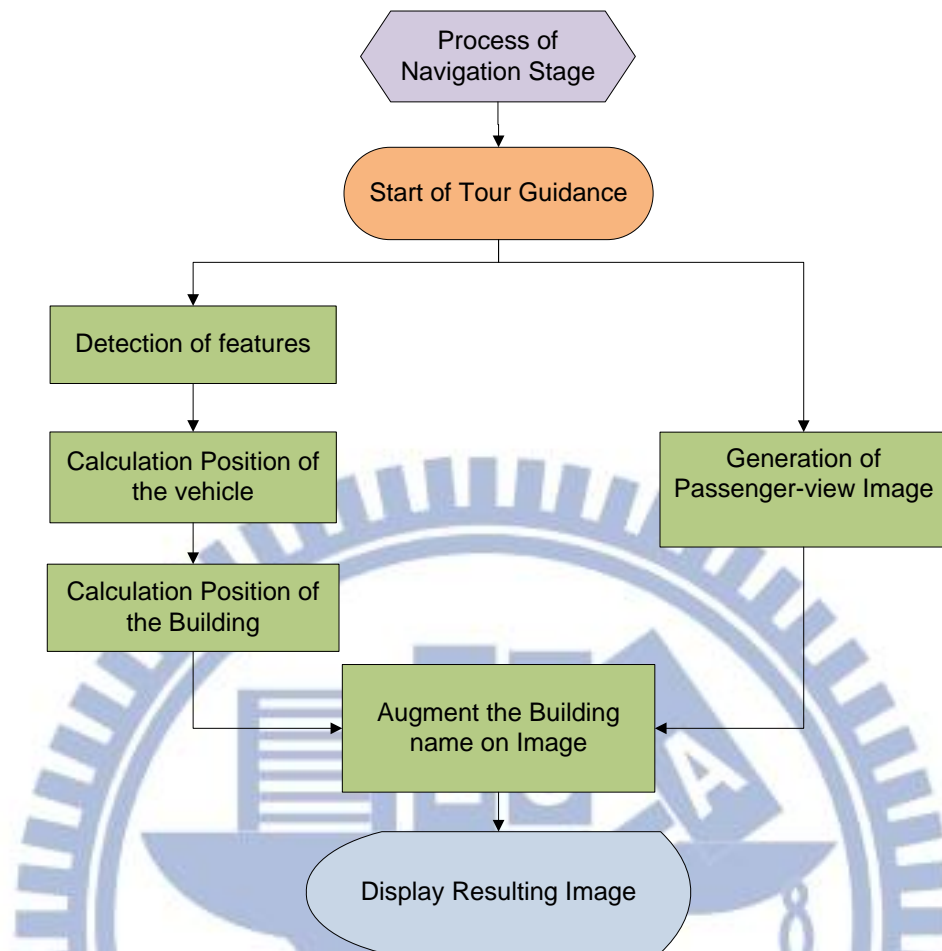


Figure 1.3 A flowchart of proposed navigation process.

1.4 Contributions

The following is a list of the major contributions made in this study.

1. A method is proposed to learn the environment map automatically.
2. A method for detecting features of vertical lines in outdoor environments using omni-camera is proposed.
3. A method for computing the position of the vehicle on a pre-selected path by detecting one or more vertical-line features in the real world is proposed.
4. A method for generating the passenger-view image by transforming the acquired omni-image and calculating the boundary position of the passenger's view is proposed.

5. A method for computing the positions of buildings on the passenger-view image, and augmenting the names of the buildings on it is proposed.
6. A tour guidance system for use on a vehicle in a park area by using AR techniques and an omni-camera is proposed.

1.5 Thesis Organization

The remainder of this thesis is organized as follows. In Chapter 2, the configuration of the proposed system and the system processes are introduced in detail. In Chapter 3, the proposed method for constructing the environment map and learning of the environment is described. In Chapter 4, the proposed method for detecting vertical-line features in the outdoor environment with an omni-camera is presented. In Chapter 5, the proposed method for vehicle localization in tours in outdoor park environments by using computer vision techniques is described. In Chapter 6, the proposed method for AR-based tour guidance is presented. In Chapter 7, experimental results and discussions are included. Finally, conclusions and some suggestions for future works are given in Chapter 8.

Chapter 2

System Design and Processes

2.1 Ideas of Proposed Method

In order to monitor the surrounding environment of the video surveillance vehicle, we choose the use of omni-cameras instead of traditional projective cameras to acquire environmental images. In this study we affix an omni-camera on top of the vehicle as shown in Figure 2.1. The omni-cameras in the device can monitor a 360-degree view of the car surround and acquire necessary scene information outside the vehicle.



Figure 2.1 The video surveillance vehicle used in this study with an omni-camera affixed on the car roof. (a) A front view of the vehicle. (b) A side view of the vehicle.

The video surveillance vehicle has high mobility so that we can move the system to everywhere. But we have to determine the best locations on top of the vehicle where the omni-imaging device should be affixed in order to enhance the imaging effect. As illustrated in Figures 2.2(a) and 2.2(b), if we affix the device at the front middle of the top of the vehicle, a half of the omni-image acquired with the

device will be occupied by the vehicle body. On the contrary, if we affix it at the right-front position on the top of the vehicle, only a quarter of the omni-image is occupied by the vehicle body instead. Therefore, in this study we decide to affix an omni-imaging device at the right-front of the top of the vehicle.

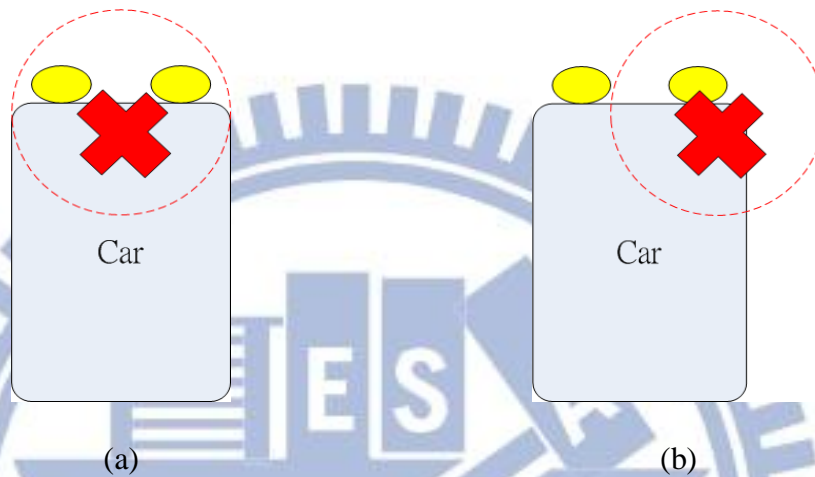


Figure 2.2 Positions of the omni-imaging device affixed to the video surveillance vehicle roof and the corresponding FOV. (a) The device is affixed at the rear-middle of the car roof. (b) The device is affixed at the right-front of the car roof.

Furthermore, we choose the upper omni-camera in the omni-imaging device for capturing features at larger heights such as those of buildings and light poles. In more detail, it is noted at first that each omni-camera captures a view of a hemisphere shape, so we can use this wide view to detect the vertical lines in the environment. In addition, a characteristic of the vertical line in the real world, when projected into the omni-image, is that the line appearing in the omni-image will go through the center of the omni-image. Therefore, we may consider vertical lines in the real world as good features and detect them in omni-images utilizing this characteristic.

Next, we localize the vehicle using *combinations* of line features. For example, at some positions along navigation paths, the omni-camera may detect the feature lines of buildings and light poles in the meantime. In such cases, we regard the

combination of the two types of features as a new type of feature learned by the system.

Moreover, we generate perspective-view images, transform them into passenger-view images seen on the mobile device held by the passenger, and augment tour-guidance information on it. Even if the back-seat passenger in the vehicle can watch the front passenger's view image on his/her own mobile device for tour guidance.

Finally, we implement the system by using a 4G/LTE network for data transmission. By using such a high-speed network, we can send the data to the server to achieve faster computations and send the results to the mobile device held by the passenger. Every passenger in the vehicle can view the tour guidance via the system. Furthermore, because we display the result on the web, even people not in the vehicle can also enjoy this AR-based tour guidance.

2.2 System of Configuration

In this section, we introduce the configuration of the proposed system in more detail. The hardware of the system includes: 1) a video surveillance vehicle, 2) an omni-camera, and 3) a laptop computer, 4) a server computer, and 5) a pad. The software includes: 1) a program used to integrate the components of the proposed system, 2) the drivers of the omni-cameras, and 3) the program for image acquisition developed by AYSIS VISION Company which is a provider of CCD cameras. The omni-camera is controlled by the laptop computer, and the pad receives information from the server computer which is kept at a cloud site.

2.2.1 Hardware Configuration

The surveillance vehicle, named Delica, is made by Mitsubishi Co. It is a vehicle with size 469cm×169cm×196cm with a working table and a power supply. System operators may sit inside the surveillance vehicle to operate the laptop computer and monitor the entire surrounding environment. Moreover, a steel frame is affixed to the top of the vehicle, on which the omni-imaging device is affixed. And extension USB cords and a cross-over cable crossing the video surveillance vehicle were added to facilitate transmitting images captured with the omni-imaging device. The entire video surveillance system is shown in Fig. 2.3.

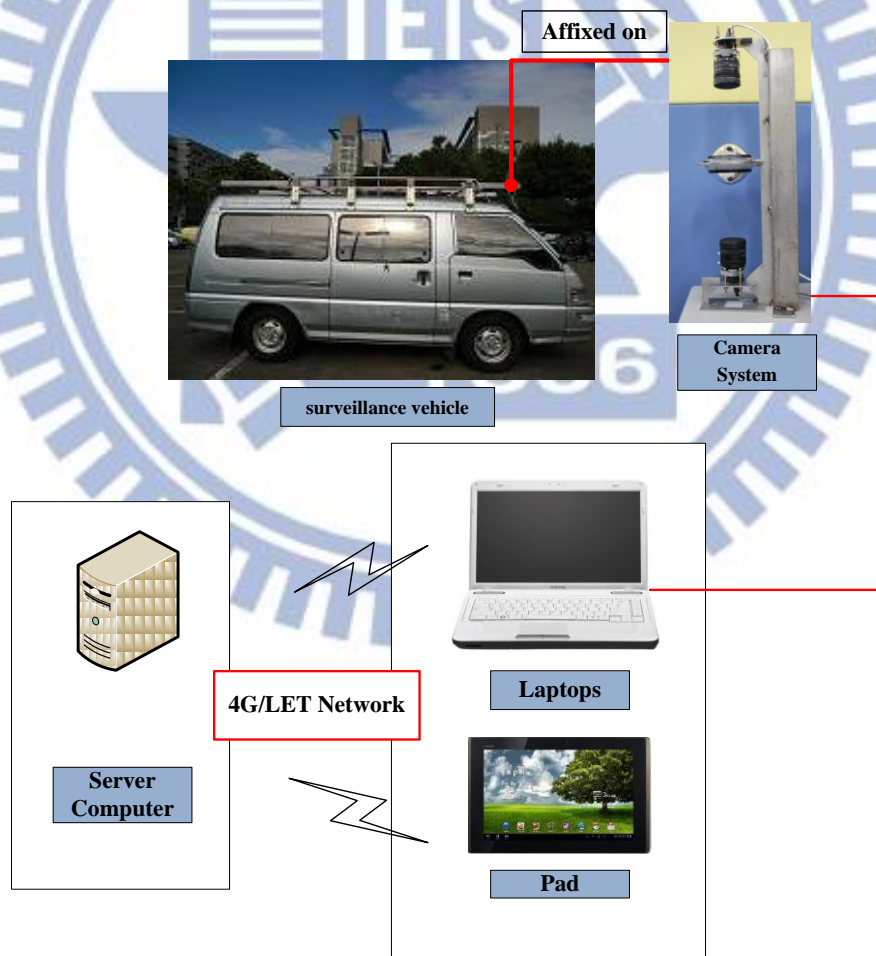


Figure 2.3 Structure of the proposed surveillance system.

In order to control the entire guidance system, we use a laptop computer, a server computer, and a pad as control units, with the laptops handling the omni-imaging device. The laptop is produced by TOSHIBA Computer, Inc. The pad, named Eee Pad Slate B121, is produced by ASUS Computer, Inc. Detailed specifications of these devices are listed in Table 2.1.

The omni-imaging device used in this study consists of two omni-cameras combined coaxially in the longitudinal direction, connected back to back, and tightened by a specially-designed steel holder. Each camera includes a lens of model JHF8M-5MP which is shown in Fig. 2.4(a), and a CMOS sensor of model AISYS ALTAIR U500C which is shown in Fig. 2.4(b). The JHF8M-5MP model is a mega-pixel lens with the parameters of 2/3", 8mm, and F2.8-22. The specification of the CMOS camera sensor is shown in Table 2.2 and the specification of the lens are shown in Table 2.3. The entire omni-imaging device shown in Fig. 2.4(c) is formed with a pair of AISYS ALTAIR U500C cameras and is affixed on the top of the steel holder.

Table 2.1 Specifications of the laptop computers and the pad used in this study.

	Satellite A660	Eee Pad Slate
CPU	Intel Core i5-480M 2.66/2.93GHz	Intel Core i5-470UM 1.33GHz
RAM	2G DDR3 1066MHz	4GB DDR3 1066MHz
GPU	ATI HD5650	None
Network	Fast Ethernet LAN	WLAN 802.11 b/g/n 2.4GHz

Table 2.2 Specification of the CMOS cameras used in the imaging device

AISYS ALTAIR U500Color Camera 5.0M	
Sensor type	CMOS
Sensor size	1/2.5" (5.70 x 4.28 mm)
Pixel size	2.2 x 2.2 μm
Frame per second	3~7 FPS
Transfer Type	USB 2.0(480 million bytes per second)

Table 2.3 Specification of the lens used in the imaging device

Lens JHF8M-5MP	
Focal Length	8 mm
Maximum Relative Aperture	1:2.8
Iris	F2.8 – 22
Angular Field of View	57.9 X 45.0 deg
Image Format	8.8 X 6.6 mm (D11mm)
Minimum Object Distance	0.1 m (From Front Vertex)



Figure 2.4 The component of the camera device and entire device. (a) AISYS ALTAIR U500C cameras. (b) JHF8M-5MP lens. (c) Entire camera device.

2.2.2 Software Configuration

We use a Visual Studio 2010 (VS 2010) as the development platform to build our guidance system. The VS 2010 is a program development tool for the operating system of Windows. The programming language we use is C++. It is a widely used language. The laptop and the pad run under the operating system of Windows 7.

In order to use the camera devices, we have to install the drivers of the ALTAIR U500C cameras into the laptop. The camera company also provides corresponding software development kits (SDKs). In addition, we can get the source codes and so we can understand the purpose of the call functions in the program. Accordingly, we can adjust the parameters of each camera, such as the value of exposure or the global color gain, through the SDK. Moreover, the camera company not only provides the VS 2010 but also the BCB, VB.NET, or C#.NET to the programmers.

2.2.3 Network Configuration

The configuration of the network used in this study is as shown in Figure 2.5, where the cameras and the laptop computer are connected through the USB cable. The computer server at the cloud site can access the images captured by the omni-cameras through the laptop by the 4G/LTE network, and so one can make sure that the system always accesses correct and immediate images and messages. Moreover, the Pad accesses the resulting images from the server computer via the 4G/LTE network also.

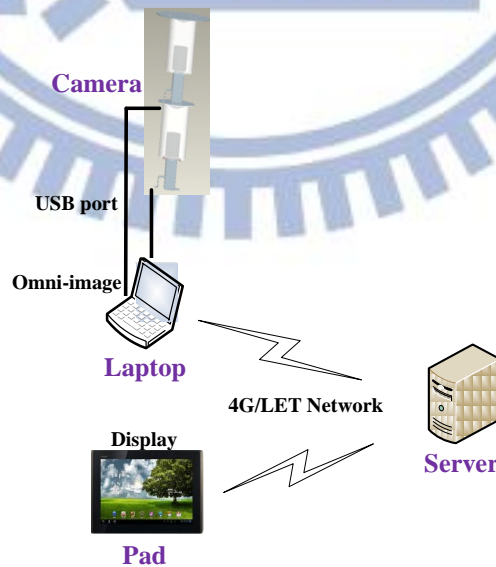


Figure 2.5 The architecture of the local network used in this study.

2.3 Network System

In this section, we describe in detail the design of the proposed network system used in this study. In Section 2.3.1, the server-side system used for conducting complicated works with long computing time in the server computer at the cloud site is described. In Section 2.3.2, we describe the client-side which includes two functions: 1) displaying the result on the pad device, and 2) sending the omni-image data to the servers. Finally, in Section 2.3.3, we will introduce the cooperative operations between the client and the server sides.

2.3.1 Server-side System

The server-side system runs on a virtual machine (VM) of the cloud server. It is connected to the laptop computer to send image data to the server, and the pad device receives the result from the server. Moreover, it has heavy computation loads while carrying out the programs implementing the proposed vision-based techniques, so we use a more powerful cloud server to implement it.

The system in the server computer gets images from the cameras on top of the vehicle. Then, it detects the line features in the images. Because the computational work is heavy, we divide the work into four parts to run them on the multi-CPU server. Next, we use the detected line features and the learned data to locate the vehicle position.

In more detail, the database of the learned information is saved in the cloud server that has lots of storage. Furthermore, the system uses the information of the location of the vehicle and the learned data to calculate the positions of the buildings in the generated perspective image, and augments the building names on it for the passenger to inspect. Finally we put the resulting image on the web site as well so that

all the users, not just the passengers inside the car, can get tour guidance by connecting to the web site.

2.3.2 Client-side System

The client-side system involves two components, namely, the laptop computer and the pad device. The laptop computer acquires the images by connecting to the omni-imaging device via the USB cable. Moreover, it transforms the omni-image into the passenger-image. Consequently, the client-side system of the laptop sends two images, which are the passenger-image and the omni-image, to the server after their resolutions are reduced to one sixteenth. Reduction of the omni-image resolution to be one sixteenth will not make the vehicle localization result wrong in most cases, but will advantageously decrease lots of data and increase the entire processing speed of the system. After all, we propose the methods that reduce the huge data into two parts; and in the meantime, the system can run continually with high vehicle localization precisions.

Finally, the pad device displays the processed images by connecting to the server and receiving processed images from it. As mentioned previously, the server system sends the processed images for AR-based guidance via the Internet, so the user can just visit the web site by using the pad device without running other applications.

2.3.3 Cooperation between Client and Server Sides

The details of the functions of the server and client side systems are described in Sections 2.3.1 and 2.3.2, respectively. Here we describe the cooperation between them in more detail. An illustration of the cooperation between the two systems is shown in Figure 2.6.

After the client-side of the laptop computer acquires an omni-image, the system transforms the omni-image into a passenger-image firstly. Then, it sends the two images, which are the passenger-image and a reduced version of the original omni-image, to the server.

After the server gets the data, it starts to analyze the data. First, the system detects the vertical line features in the reduced omni-image, and matches them with the learned feature data to localize the vehicle. Next, the system calculates the positions of the building and augment its name on the passenger-view image. Finally, the system sends the AR result to the client-side device via the web site. That is, the client-side uses the pad device to displays the AR result by visiting the web site so that any user in the vehicle can enjoy the guidance by using this system.

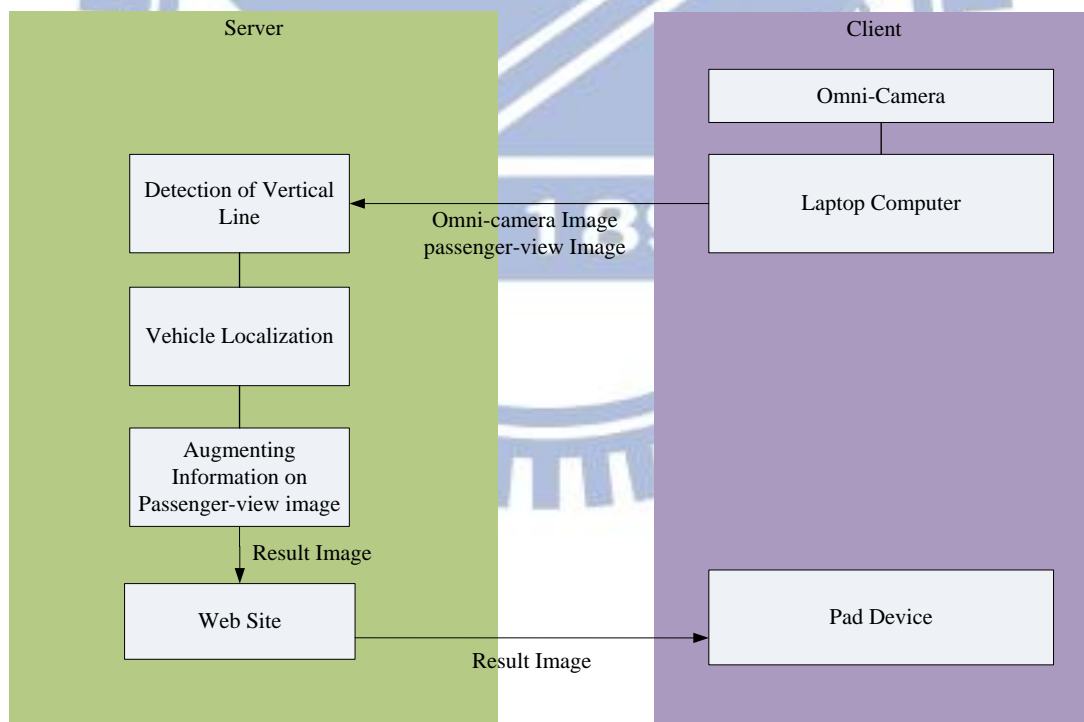


Figure 2.6 Cooperation between client and server sides.

2.4 System Processes

2.4.1 Learning Process

The first part of our guidance system is the learning process. It plays an important role in our system. At the beginning, we need a real-world environment map. Therefore, we choose the open-source map on the Internet to construct the first part of the environment map. Next, we choose a path on the environment map and define it. Then, we save the result into the database.

Next, we equip the omni-camera on the top of the vehicle and drive it on the path we choose. When travelling on the path, the learning system detects vertical lines “seen” in the images acquired with the omni-camera in the meantime. The operator can see the vertical lines detected by system on the laptop computer, and then he/she makes the vehicle stop and starts to “learn” the features. The learned information includes two types: are building and light pole. The system saves information, such as the type of each feature (building or light pole), the position of the feature in the map, the angle of the feature, etc., as the learning result into the database. The detail of the feature information will be discussed in Chapter 3. After traveling the path, the system analyzes the information automatically. In this process, the system works like simulating the traveling once again, and saves the analyzed data in the form of tables which may be looked up in the navigation process to speed up the guidance system. In more detail, the system can use the tables to match the detected features while running the navigation system rather than to analyze it again and again. A flowchart of the above learning process is shown in Fig. 2.7.

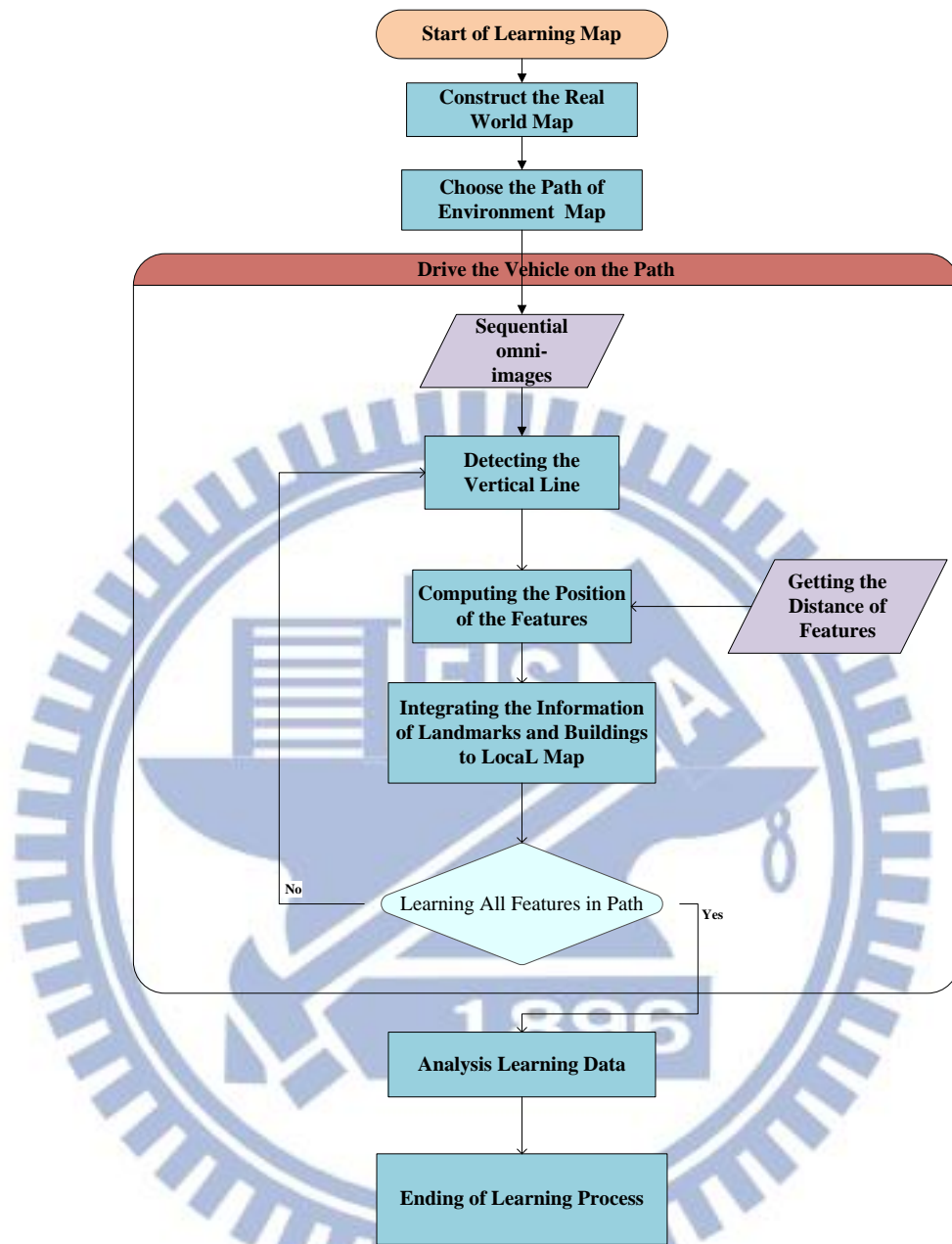


Figure 2.7 Flowchart of learning process.

2.4.2 Navigation Process

The second part of the proposed guidance system is the navigation process. In Section 2.4.1, we mentioned how we learn about environment through the learning process. Accordingly, we can estimate the position of the vehicle on the environment

map and implement our tour guidance system in the navigation process. First of all, we use the captured omni-image to detect the vertical line-shaped objects in the environment. This process will be introduced elaborately in Chapter 4. Then, by using the learning information and the detected features, the system can localize the current position of the vehicle in the environment map. The detailed process will be introduced in Chapter 5. Next, the system can calculate the position of the building and augment the building information on the passenger-view image. The detailed process will be introduced in Chapter 6. The entire navigation process is shown as a flowchart in Figure 2.8.

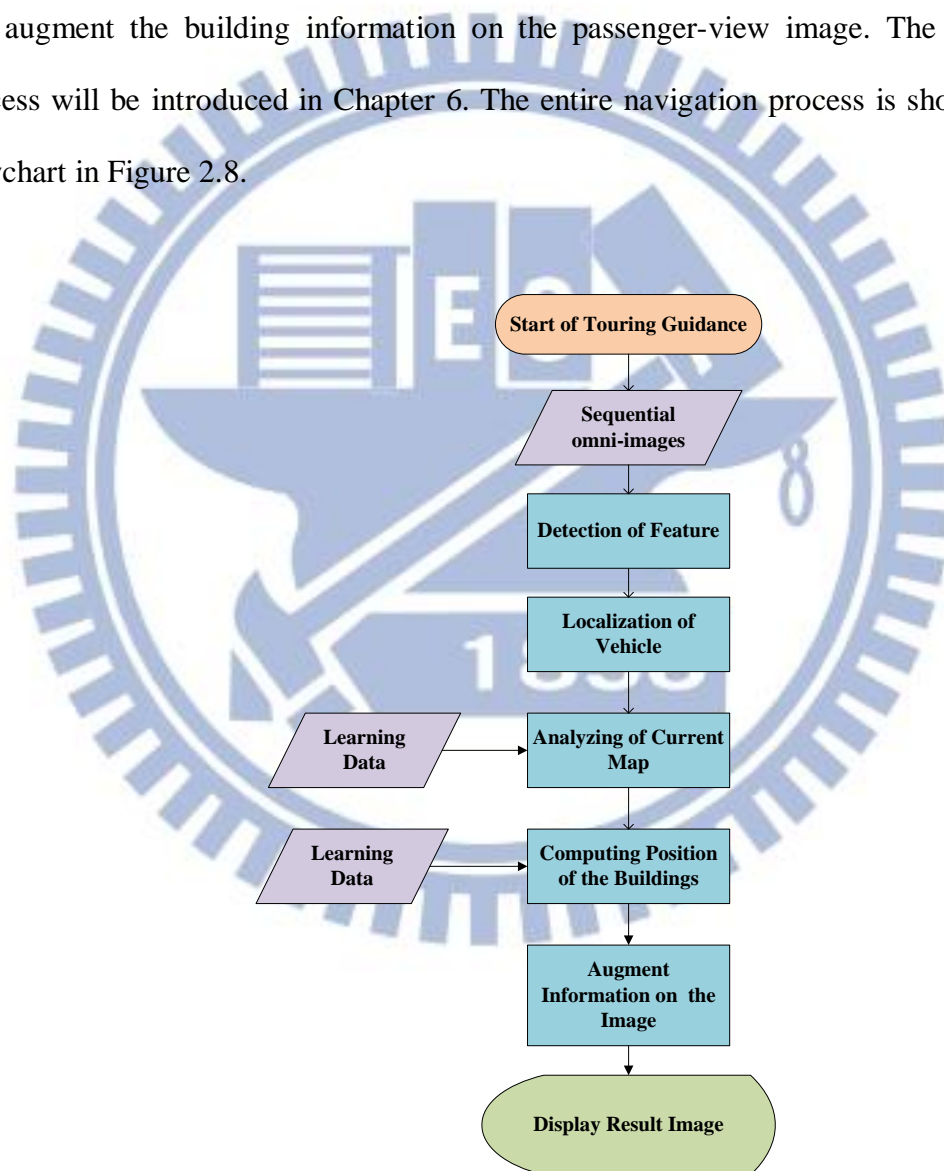


Figure 2.8 Flowchart of proposed tour guidance system.

Chapter 3

Learning of Environments

3.1 Ideas of Proposed Environment Learning Techniques

In this chapter, we describe the details of the method we propose to generate the environment map for use in the proposed AR-based tour guidance system. In order to complete the system, we must construct the environment map for use in the navigation phase, which includes the information about the path of the tour, the line feature detected for vehicle localization, and the building information.

The first part of environment learning is the construction of a real-world map. We choose the “OpenStreetMap” to construct our environment map. The OpenStreetMap is an open data commons where peoples can modify the map free on the internet like Wikipedia. We use the real-world map acquired from there as the base of the environment map for this study, and define features of the environment for my system. In more detail, each feature we define will be marked with an icon on the map. The selected path is also marked on the map. In addition, the system can also show the vehicle position on the map during the tour so that the user can see the map clearly.

Next, we learn the line-shaped features for vehicle localization. In order to make our system more accurate, we have to get more information about the features. In addition, we not only learn the position of each feature on the map but also learn the information of the feature about how the camera on the vehicle can “see” It. The detail will be described in Section 3.3.

Furthermore, we learn the building information for the system to show in the AR image. The building information includes the building name, the building area in the map, the area where the camera can see, and so on.

Finally, we merge all the data of the environment as the environment map that the system can use for navigation in the tour. The detail of environment learning will be described in the following.

3.2 Coordinate Systems Used in This Study

In this section, we will introduce the coordinate systems used in this study, which describe the relations between the used devices and the environment map. The following are the four coordinate systems used in this study.

- (1) *World coordinate system (WCS)*: denoted as (x, y, z) as shown in Figure 3.1(a). The origin O_w of the WCS, a pre-defined point on the ground, is regarded as the starting position of the path traversed by the vehicle during the learning and navigation processes.
- (2) *Camera coordinate system (CCS)*: denoted as (X, Y, Z) as shown in Figure 3.1(b). The origin O_m of the CCS, a focal point of the hyperboloidal-shaped mirror, lies on the X - Y plane which is coincident with the image plane. The Z -axis coincides with the optical center of the lens of the upper CMOS camera in the omni-imaging device.
- (3) *Image coordinate system (ICS)*: denoted as (u, v) as shown in Figure 3.1(c). The u - v plane of this system coincides with the image plane with the origin O_c located at the center of the image plane.
- (4) *Map coordinate system (MCS)*: denoted as (M_x, M_y) as shown in Figure

3.1(d). The MCS is used to represent the environment map. The M_x-M_y plane coincides with the image plane of the floor. The origin is at the left-top position of the image plane.

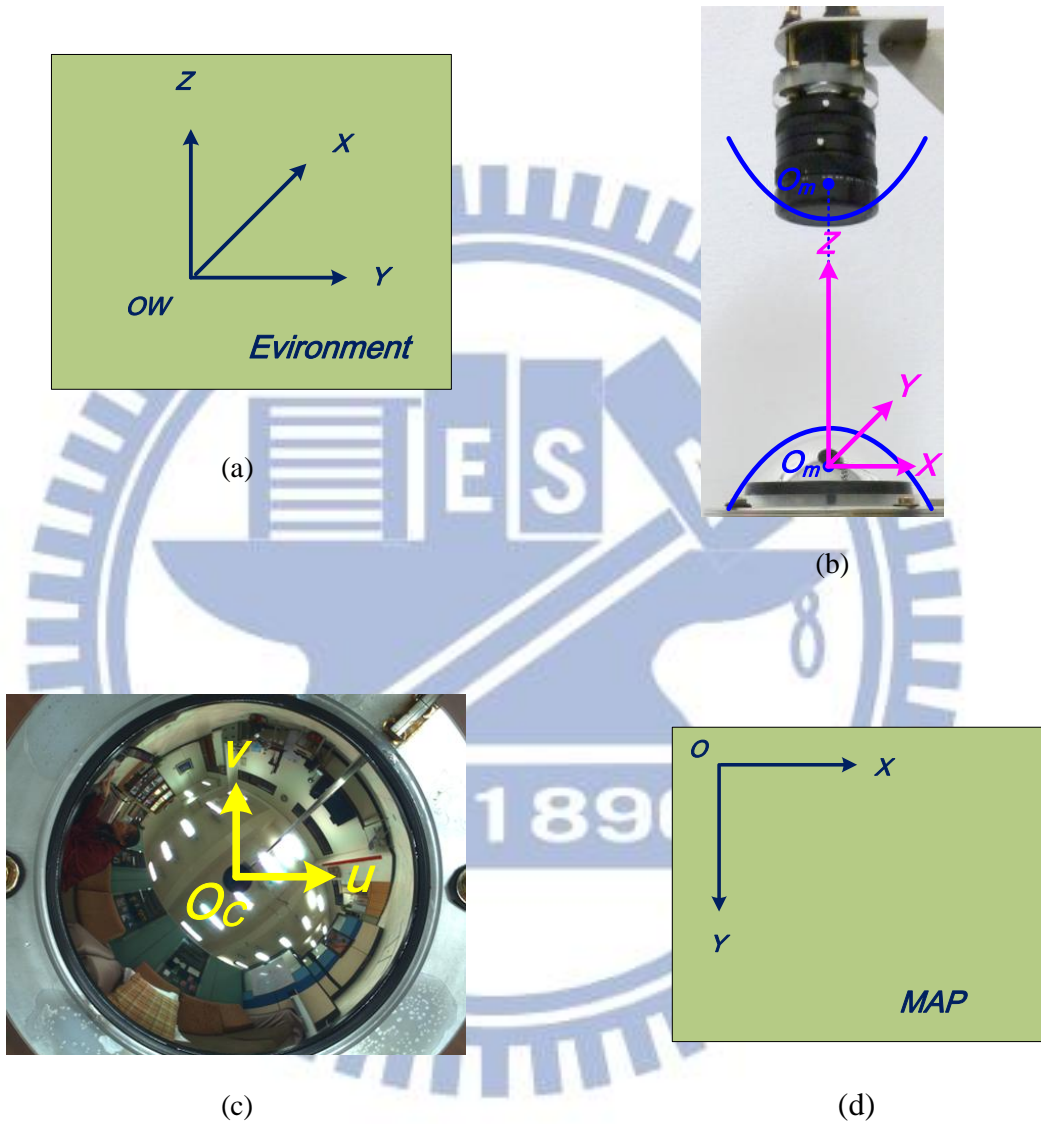


Figure 3.1 The three coordinate systems used in the proposed system. (a) The global coordinate system. (b) The camera coordinate system. (c) The image coordinate system.

3.3 Construction of Environment Map

In this section, we will introduce the method we propose to construct the

environment map. The environment map is like a database, which contains the information that we use in the navigation process.

3.3.1 Information Included in Environment Map

The information put in the environment map includes the real-world map, the navigation path, the vertical lines in environment, the building information. The real-world map is constructed by the use of the “OpenStreetMap” in a website, as shown in Figure 3.2. It is constructed by the geometry and the text describing the buildings. In addition, we define a path on the map using *piecewise* line segments. The information of the path includes the positions of the end points of the path and the length of it. Moreover, the features for matching along the path are also learned.

Next, we define the features for each line segment, which includes its position on the map, the relation of the path and the feature, and the angle between the feature and the road. Finally, building information for showing the AR image is learned, which includes the area of the building, the name of the building, the relation between the building and the path.

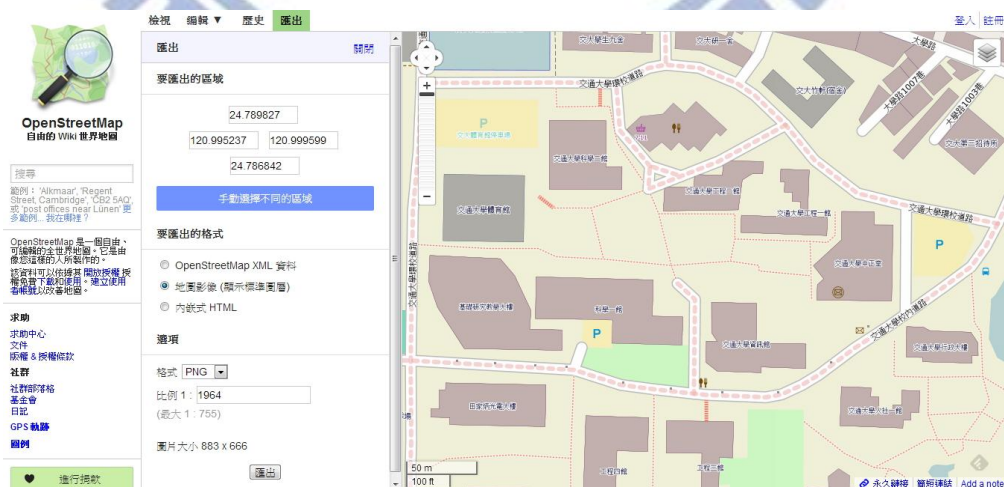


Figure 3.2 User interface for real-world map construction by use of OpenStreetMap.

3.3.2 Creation of Database for Environment Map

In order to create a database for use by the system in the navigation process, we have to save the data orderly. First, we construct a 2-dimension (2D) environment map by the “OpenStreetMap” and choose a part of the area on the map which is big enough to include the park environment. Next, we set an appropriate scale and a data type for the map. In this study, the scale of the map is 1 centimeter to 10 meters, and the data type of the map is set to be “.jpeg” as shown in Figure 3.3.

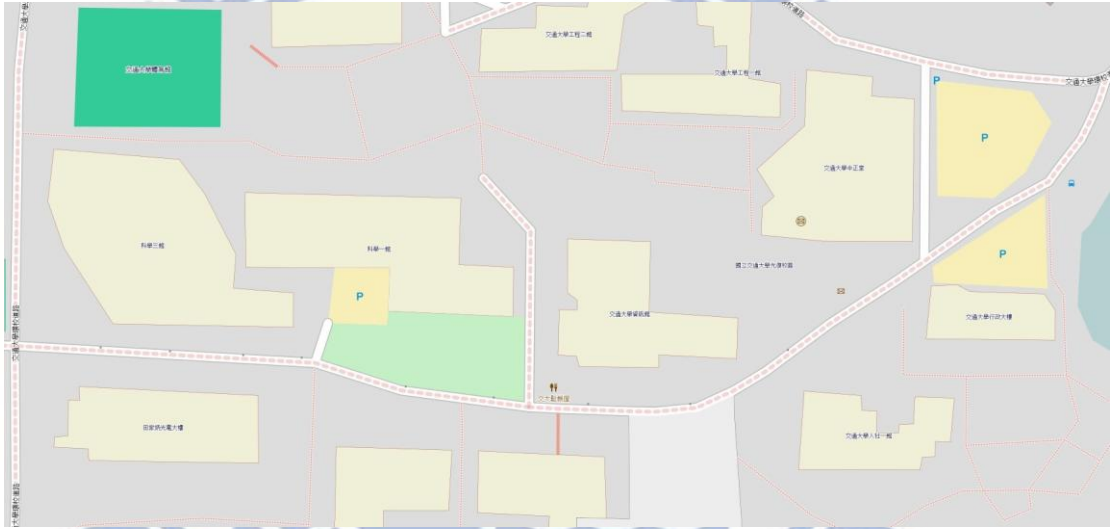


Figure 3.3 The real-world map we use in the proposed system.

Furthermore, we save the data of the path into a data structure we designed. Specifically, we divide the path into some line segments, each being represented by two points. Moreover, the length of any line segment is also calculated and saved in the database. The length $dist_p$ of each line segment p with end points at coordinates (x_1, y_1) and (x_2, y_2) is calculated by the following equation:

$$dist_p = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} . \quad (3.1)$$

In addition, the orientation θ_p of the line segment p is also calculated and saved, which is calculated by the following equation:

$$\theta_p = \tan^{-1}\left(\frac{x_1 - x_2}{y_1 - y_2}\right). \quad (3.2)$$

where the orientation θ_p means the angle between the line segment and the horizontal direction in the map.

The subsequent major task is to save the information about the features which match with the path. In more detail, we save the number of features matched and the tag of each feature we defined.

Also, the system has to “learn” as well the vertical lines for vehicle localization. For this, it saves the position of each vertical line found in the environment as a point in the map. Also, while the vehicle is moving on the path, only on a part of the path can be “seen” by the cameras on the vehicle; therefore, for each detected vertical line, we save the orientations in which the first time and the last time the system in the vehicle can detect the vertical line. In other words, we learn the range of the views in degrees in which the vertical line can be seen.

Finally, the system learns the building for showing the AR image. For this, the information of the building we save into the system includes the geometry of the building drawn by lines, the building name, and the tag of the building we defined.

3.4 Learning of Environment Features

In this section, we introduce the proposed methods to learn the information about the environment. The learning process is a necessary step for the system to get ready to run.

3.4.1 Learning of Navigation Paths

A path for car driving in the real world is not just a straight line, but in the study, we use piecewise line segments to describe a path. Furthermore, the system “sees”

each line segment as a unit which includes a lot of information about it. The detail of learning a navigation path is described in the following algorithm.

Algorithm 3.1 Learning of a navigation path.

Input: A real world map.

Output: An environment map with line segments drawn on it as a selected path and the information of the path in a type of data structure.

Steps:

- Step 1. Choose a line segment which can be used to compose a desired path by defining the start point and the end point on the map for the line segment.
- Step 2. Calculate the length of the line segment by using Equation (3.1) and save it into the data structure of the path.
- Step 3. Calculate the angle of the line segment by using Equation (3.2) and save it into the data structure of the path.
- Step 4. Add all the features that the vehicle driving on the line segment of the path can “see,” and save the number of the features also, into the data structure of the path.
- Step 5. Repeat Steps 1 to 4 until all the line segments of the desired path are chosen.
- Step 6. Draw all the line segments as a path on the map.

It noted that the action of adding the features in Step 4 will be described in more detail in Section 3.4.2. After learning the path, the system can use the path data together with other learned data to navigate in the environment.

3.4.2 Learning of Vertical Lines in Environments

The learning of vertical lines includes two parts. The first part is to learn a line feature at a time. The second is for the system to learn multiple line features simultaneously.

It is noted that learning the path should be conducted before learning the vertical line features. Furthermore, the most important step in our system is to calculate the angle of the feature orientation on the map. In more detail, an illustration showing the vehicle on the road is shown in Figure 3.4(a). And an illustration of a detected feature, a light pole, is shown in Figure 3.4(b). We can easily calculate the angle of the orientation of the feature as can be seen from the illustrations. The following algorithm describes the first part of the proposed learning process — learning of vertical-line features along the selected path.

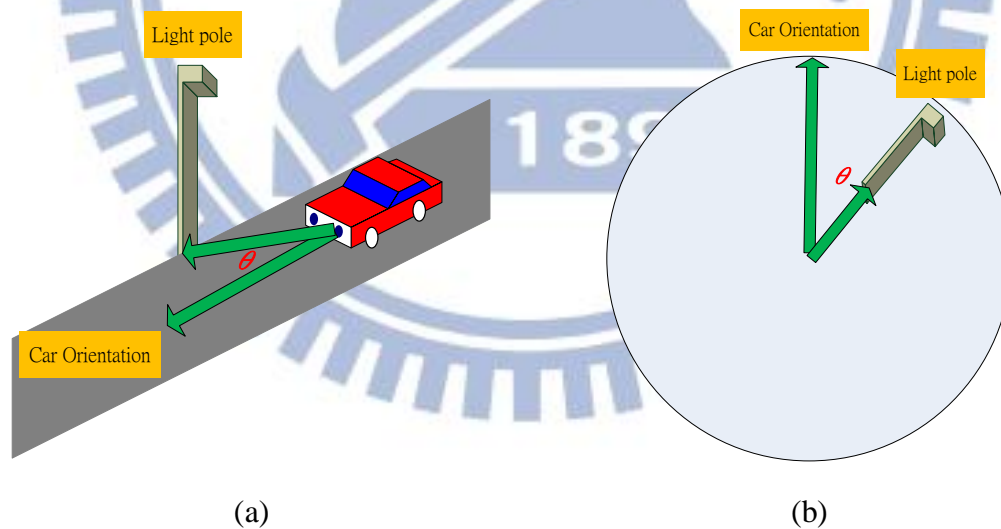


Figure 3.4 The vehicle on a path while detecting a feature. (a) An illustration of the vehicle driving on the path. (b) An omni-image with a detected feature — a light pole.

Algorithm 3.2 *Learning of vertical line features.*

Input: A real-world map with the path information and an omni-image I_1 acquired by

with the omni-imaging device on the vehicle.

Output: An environment map with the along-path features drawn on it and the information of the features in a type of data structure.

Steps:

Step 1. Drive the vehicle along each line segment l_i along the path and detect the vertical lines in image I_1 using the feature detection method described in Chapter 4.

Step 2. While the vertical lines are being detected, stop the vehicle and do the following steps.

2.1 Measure the position of the feature on the map using the scale of the map and save it into the environment map by associating the feature with the corresponding line segment l_i of the path.

2.2 Compute the orientation by which the system detects the feature for the first time, call it the *first-angle* of the feature, and save it into the environment map.

2.3 Drive the vehicle forward until the feature can no longer be detected by the system.

2.4 Compute the orientation by which the system detects the feature for the last time, call it the *last-angle* of the feature, and save it into the environment map.

Step 3. Repeat Steps 1 and 2 until the vehicle arrives at the end point of the path.

Step 4. Draw all the features on the map.

Next, learning multiple features simultaneously is different from learning one at a time. The learning of multiple features conducted in this study is a new method. For vehicle localization using multiple line features, we propose to use the longest

common subsequence (LCS) algorithm in this study, which is based on the dynamic programming technique. It enables the vehicle to drive on the path by using the data learned from Algorithm 3.2 and the path information.

In some cases, the vehicle on the path may detect many features at a time like the case illustrated in Figure 3.5(a). And the system has to learn these multiple features by saving all the angles of the features as illustrated in Figure 3.5(b). In more detail, we calculate the orientation of a feature by the following equation:

$$\theta_f = \theta_g - \theta_p \quad (3.3)$$

where θ_p means the angle between the line-segment path and the horizontal direction in the map that is learned by Algorithm 3.1; θ_g is the feature direction with respect to the horizontal direction in the map; and θ_f is the feature direction with respect to the path direction as shown in Figure 3.5(c). Then, we save the value of θ_f into the learning data.

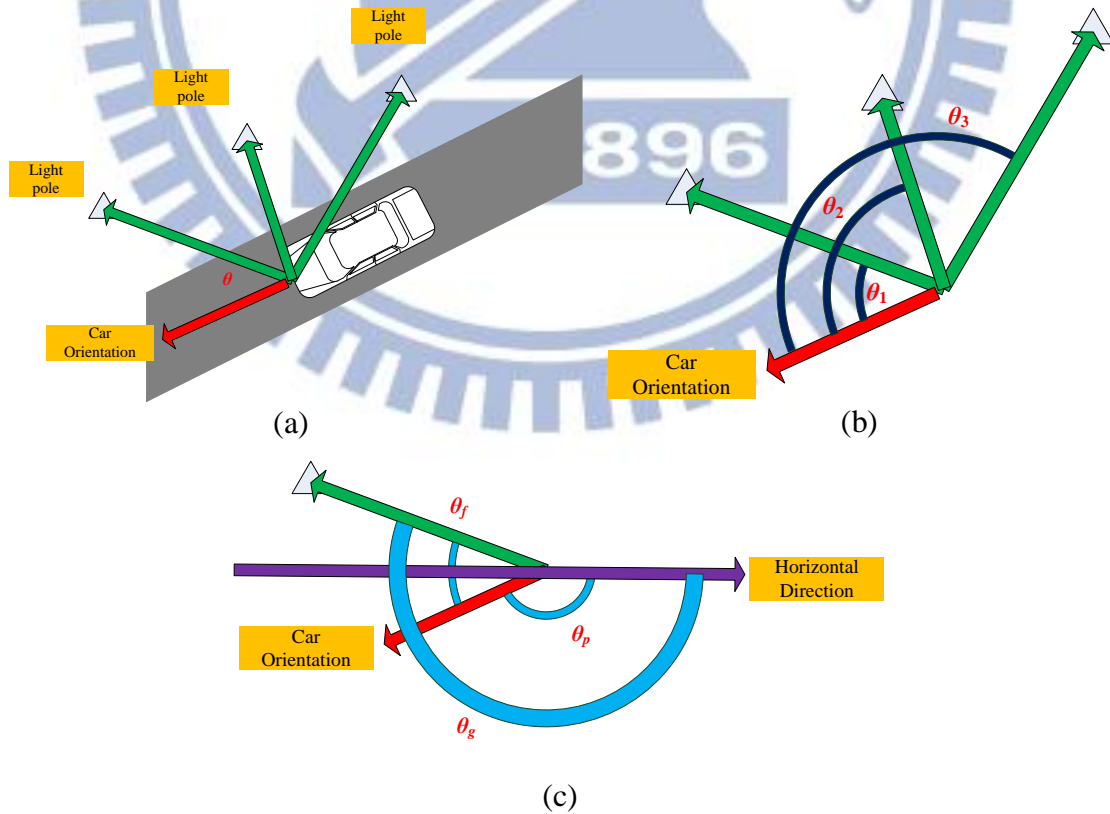


Figure 3.5 Illustrations of multiple feature detection. (a) Illustration of detected feature

on the map. (b) Detected angle of features. (c) Illustration of calculating the angle of θ_f .

We use the detected line features to localize the vehicle by matching the angles of the features using the LCS algorithm to. The details are described in Chapter 5. In the following, the algorithm for learning multiple line feature data to localize the vehicle is described.

Algorithm 3.3 Learning of multiple line features.

Input: A real-world map with the path and feature information.

Output: A table T of the multiple line feature data.

Steps:

- Step 1. Start with the first line segment l of the pre-selected path in the map.
- Step 2. Start with the first point/pixel of line segment l .
- Step 3. Calculate the angle between l and each feature which matches l by do the following steps.
 - 3.1 Choose the feature associated with l whose data are learned from Algorithm 3.2.
 - 3.2 Calculate the feature angle by using Equations 3.2 and 3.3 which setting the point/pixel of the vehicle and the point/pixel of the feature as two end points.
 - 3.3 Save the result into table T .
 - 3.4 Repeat Steps 3.1 to 3.3 until all the features matching l are learned.
- Step 4. Move to the next point/pixel along the line segment.
- Step 5. Repeat Steps 3 and 4 until the end point/pixel of the current segment is reached.
- Step 6. Repeat Steps 2 to 5 until the end line segment of the path is reached.

3.4.3 Learning of Building Information

Since the system must show the building information in the AR image, an algorithm for learning the building is necessary and is derived in the following. An illustration of the algorithm is shown in Figure 3.6(a). The learning of the building corner is shown in Figure 3.6(b). In more detail, we learn the edge line of the building by connecting two corners. And it noted that the edge line we learn is the side which the vehicle can “see”. In other words, there is no need to learn the side which can’t be seen when driving on the path. Then, the system can calculate the position of the building while driving on the path by the result of learning the edge line segments. The details of showing the AR image will be described in Chapter 6.

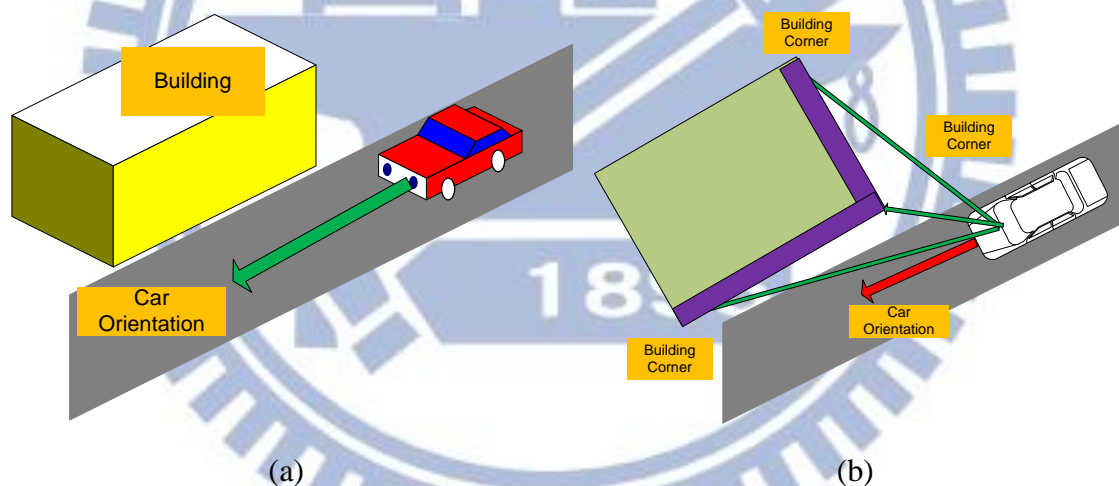


Figure 3.6 Learning of buildings. (a) An illustration of learning the building. (b) The result of learning the building in the map.

Algorithm 3.4 Learning of building information.

Input: A real-world map image with the path information.

Output: A table of the data of learned buildings.

Steps.

- Step 1. Drive the vehicle on the path and detect the building(s) in the acquired image.
- Step 2. If a building is detected, stop the vehicle and do the following steps.
- 2.1 Save the position of the building corner P_i on the map by measuring it and using the scale of the map to calculate the position of P_i on the map.
 - 2.2 Repeat Step 2.1 until all building corners P_i are learned.
 - 2.3 Define a line by connecting every two corners P_i .
 - 2.4 Save the building name in the data structure.
- Step 3. Repeat Steps 1 and 2 until the vehicle arrived at the end point of the path.

3.5 Experimental Results

The map of our experimental environment is shown in Figure 3.7. It is the final result of learning all features and the path. In more detail, we will present the results of applying the proposed algorithm in this chapter step by step. First, in the part of learning the navigation path, we defined a path in the environment as shown in Figure 3.8. After learning the path, we have a lot of information about it. Next, in the part of learning vertical line features, we define points of features in the environment as shown in Figure 3.9. Furthermore, the different colors of the features represent different kinds of them, such as light pole and edge line on building walls, etc. After the learning processes are completed, the system can use the resulting information to locate the vehicle. Finally, in the part of learning building information, we show the result in Figure 3.7. All the learned data are saved in the system in many data structures we defined. Before the system starts to run, the data will be loaded into the

system.

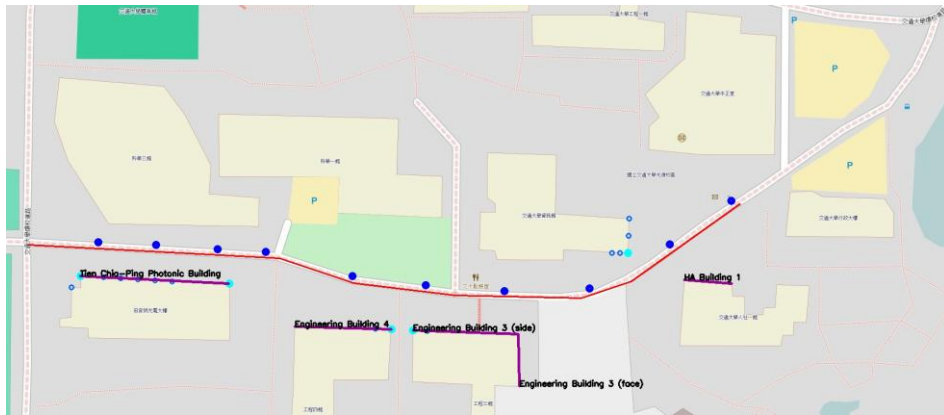


Figure 3.7 The environment map we use in the proposed system.

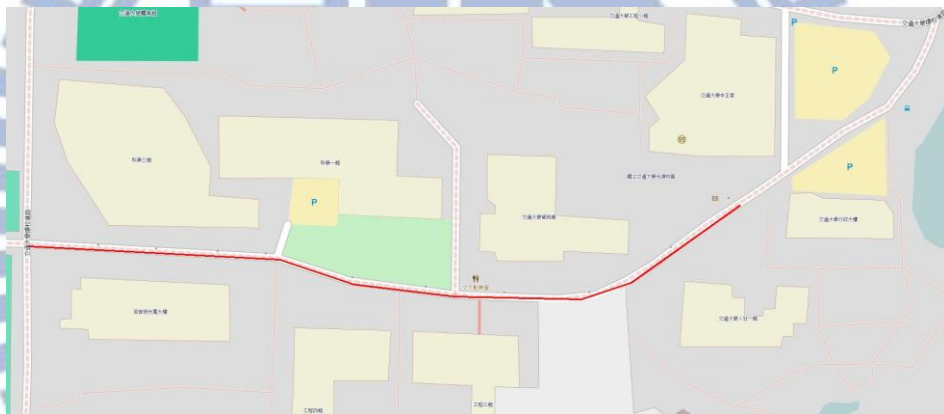


Figure 3.8 The environment map with the path.



Figure 3.9 The environment map with the path and features.

Chapter 4

Automatic Detection of Vertical Lines in Environments with an Omni-camera

4.1 Introduction

In this chapter, we describe the proposed method for detecting vertical-line features in omni-images around the vehicle. Vertical-line features include light poles on street sidewalks or edge lines on building walls. Sometimes, such features can even be tree trunk or some non-artificial object. In other words, all the objects with the shapes of vertical lines can be features for use in our system. So our system can match most application environments.

At first, our main idea of vertical-line feature analysis is that a vertical line in real world, when mapped into the omni-image, becomes a radius line in the image. In more detail, the object we see in the omni-image will generally be distorted. But only the vertical line in the omni-image will not be so. So we take advantage of this characteristic and regard the vertical line as a feature for use in our study.

Secondly, a vertical line in the real world is not just a line in the mathematical meaning; in other words, it has a certain width and may not be totally straight everywhere on the line. It might be a broken line as well. So it has a lot of problems to be solved before it can be well detected in omni-images by a software program. In this study, we propose many techniques to solve the problems and a method using the

techniques to detect vertical line features in the real world successfully.

The idea behind the proposed method will be described in Sections 4.2, and the related techniques and an algorithm to implement the method will be described in Section 4.3. The algorithm is designed mainly for localizing the vehicle position in the outdoor environment. And while the vehicle is driven in an outdoor environment, the system uses the algorithm to detect vertical-line features which can be seen along the pre-selected path.

Finally, some experimental results will be shown in Section 4.5, including some figures and descriptions about the result.

4.2 Idea of Analysis of Vertical Lines in Omni-images

In this study, we use vertical line as a feature to localize the vehicle. In addition, the vertical line in the real-world space becomes a radial line in the omni-image. And this fact has been provided by Wu and Tsai [8]. It is reviewed subsequently.

As mentioned previously, the hyperbolic shape of the mirror in an omni-camera may be described as:

$$\frac{R^2}{a^2} - \frac{z^2}{b^2} = -1, \quad R = \sqrt{x^2 + y^2}, \quad c = \sqrt{a^2 + b^2}. \quad (4.1)$$

As depicted in Figure 4.1, the *omni-camera* and *omni-image coordinate systems* are specified by coordinates (x, y, z) , and (u, v) , respectively; and the projection relationship between the *omni-image coordinates* (u, v) and the *omni-camera coordinates* (x, y, z) can be described as follows [13]:

$$u = \frac{xf(b^2 - c^2)}{(b^2 + c^2)(z - c) - 2bc\sqrt{(z - c)^2 + x^2 + y^2}},$$

$$v = \frac{yf(b^2 - c^2)}{(b^2 + c^2)(z - c) - 2bc\sqrt{(z - c)^2 + x^2 + y^2}}, \quad (4.2)$$

where the parameter f is the focal length of the omni-camera.

According to [14][15], the relation between the coordinates (X, Y, Z) of a space point P and the image coordinates (u, v) of its corresponding projection point p in the image may be described by

$$\tan \alpha = \frac{(b^2 + c^2) \sin \beta - 2bc}{(b^2 - c^2) \cos \beta}; \quad (4.3)$$

$$\cos \beta = \frac{r}{\sqrt{r^2 + f^2}}; \quad (4.4)$$

$$\sin \beta = \frac{f}{\sqrt{r^2 + f^2}}; \quad (4.5)$$

$$\tan \alpha = \frac{Z - c}{\sqrt{X^2 + Y^2}}, \quad (4.6)$$

where $r = \sqrt{u^2 + v^2}$ and f is the camera's focal length. We assume that a, b, c , and f are known in advance. Also, according to the *rotational invariance* property of the omni-camera [4], we have

$$\cos \theta = \frac{X}{\sqrt{X^2 + Y^2}}; \quad (4.7)$$

$$\sin \theta = \frac{Y}{\sqrt{X^2 + Y^2}}, \quad (4.8)$$

$$\cos \theta = \frac{u}{\sqrt{u^2 + v^2}}; \quad (4.9)$$

$$\sin \theta = \frac{v}{\sqrt{u^2 + v^2}}, \quad (4.10)$$

where θ is *both* the angle of space point P with respect to the X -axis, and that of image point p in the image coordinate system with respect to the u -axis. The above equations may be used to derive the relation between (u, v) and (X, Y, Z) .

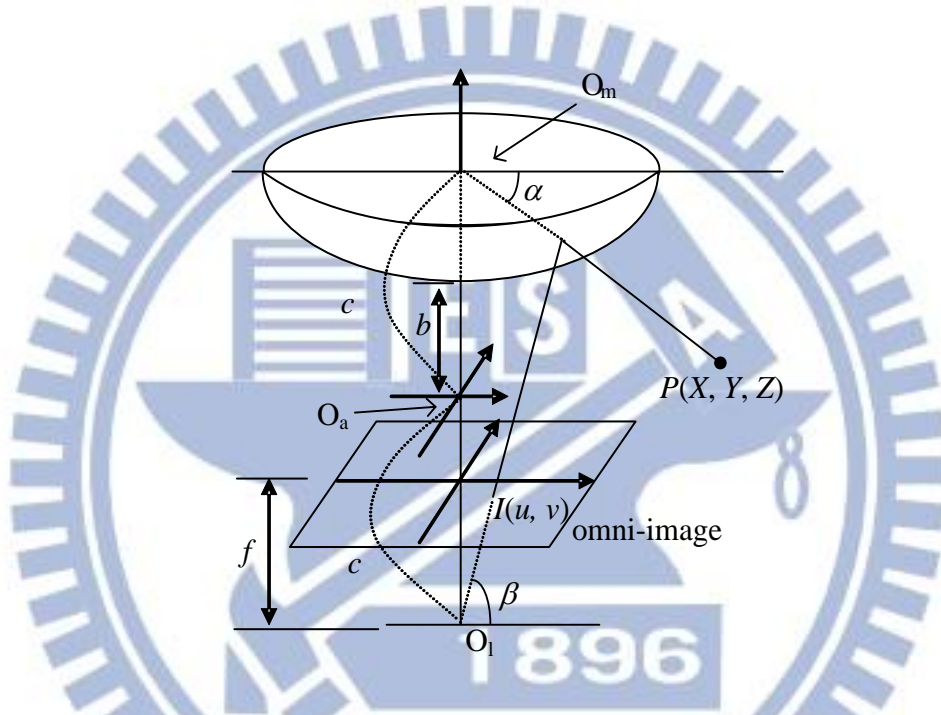


Figure 4.1 Camera and image coordinate systems.

As shown in Figure 4.2, given a space line L with an end point P_0 with camera coordinates (X_0, Y_0, Z_0) , any point P on L with camera coordinates (X, Y, Z) and point P_0 together form a vector $V_0 = (X - X_0, Y - Y_0, Z - Z_0)$. On the other hand, let the direction vector of L be denoted as $V_L = (d_x, d_y, d_z)$. Then, since V_0 and V_L are parallel, we get the equality $V_0 = \lambda V_L$, or equivalently,

$$(X, Y, Z) = (X_0 + \lambda d_x, Y_0 + \lambda d_y, Z_0 + \lambda d_z) \quad (4.11)$$

where λ is a parameter. Also, let S be the space plane going through line L and the

mirror base center O_m at camera coordinates $(0, 0, c)$, and let $N_S = (l, m, n)$ be the normal of S . Then, any point P' at camera coordinates (X, Y, Z) on S and point O_m together form a vector $V_m = (X - 0, Y - 0, Z - c) = (X, Y, Z - c)$ which is perpendicular to N_S so that the inner product of V_m and N becomes zero, leading to the following equality:

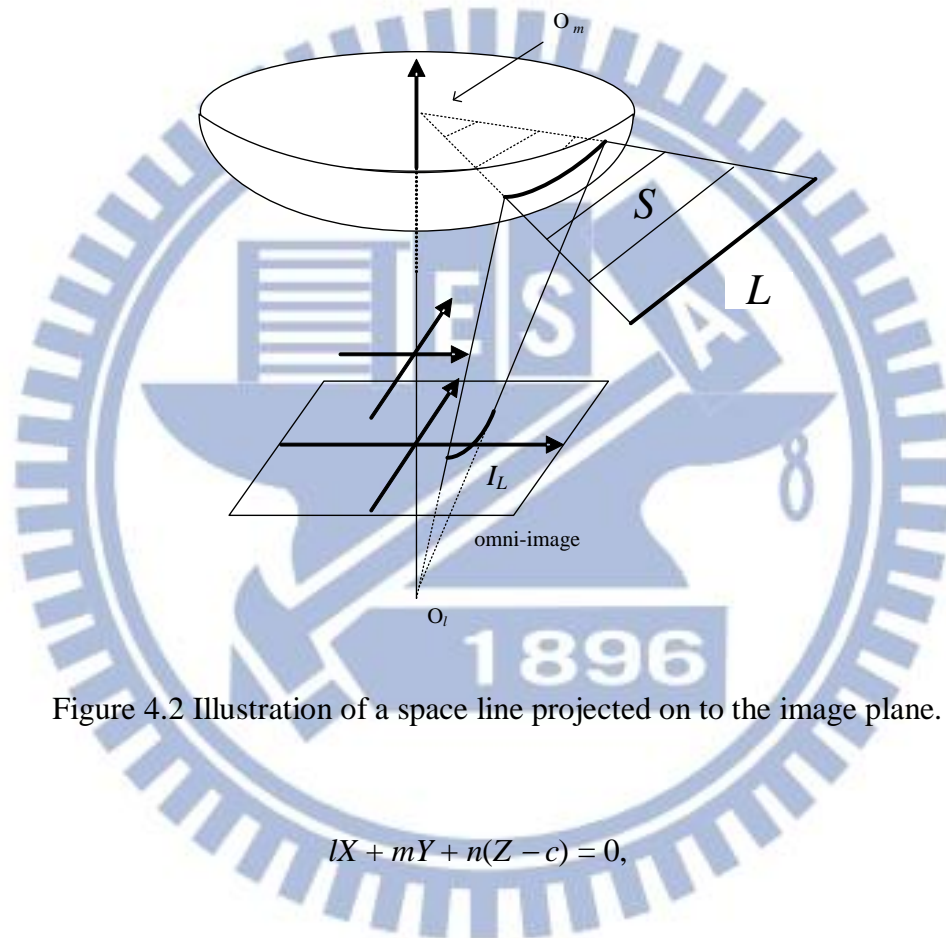


Figure 4.2 Illustration of a space line projected on to the image plane.

$$lX + mY + n(Z - c) = 0, \quad (4.12)$$

or equivalently,

$$Z - c = -\frac{lX + mY}{n}. \quad (4.13)$$

Now we want to derive the equation of the projection of space line L on the image, which expresses the relation between the camera coordinates (X, Y, Z) of a space point P' on L and the image coordinates (u, v) of the image point p'

corresponding to P' . Note that P' is also on plane S . Combining (4.6) through (4.10) and (4.13), we get

$$\begin{aligned}\tan \alpha &= \frac{Z-c}{\sqrt{X^2+Y^2}} = -\frac{l\frac{X}{\sqrt{X^2+Y^2}} + m\frac{Y}{\sqrt{X^2+Y^2}}}{n} \\ &= -\frac{l\frac{u}{\sqrt{u^2+v^2}} + m\frac{v}{\sqrt{u^2+v^2}}}{n} = -\frac{\left(\frac{lu}{n} + \frac{lv}{n}\right)}{\sqrt{u^2+v^2}}.\end{aligned}\quad (4.14)$$

On the other hand, substituting (4.4) and (4.5) into (4.3), we get

$$\tan \alpha = \frac{(b^2+c^2)\frac{f}{\sqrt{f^2+r^2}} - 2bc}{(b^2-c^2)\frac{r}{\sqrt{f^2+r^2}}} = \frac{(b^2+c^2)f - 2bc\sqrt{f^2+u^2+v^2}}{(b^2-c^2)\sqrt{u^2+v^2}}.\quad (4.15)$$

Equating (4.14) and (4.15) leads to

$$\frac{lu}{n} + \frac{mv}{n} + \frac{(b^2+c^2)f}{(b^2-c^2)} = \frac{2bc\sqrt{f^2+u^2+v^2}}{(b^2-c^2)}$$

which may be squared and reduced to get the following desired result:

$$(A^2 - D^2)u^2 + 2ABuv + (B^2 - D^2)v^2 + 2ACu + 2BCv + E = 0\quad (4.16)$$

where

$$A = \frac{l}{n}, \quad B = \frac{m}{n}, \quad C = \frac{(b^2+c^2)f}{(b^2-c^2)}, \quad D = \frac{2bc}{(b^2-c^2)}, \quad E = C^2 - D^2f^2.\quad (4.17)$$

Multiplying (4.16) by n^2 , we can get an alternative form of (4.16) without A and B as follows:

$$(l^2 - n^2D^2)u^2 + 2lmuv + (m^2 - n^2D^2)v^2 + 2lnCu + 2mnCv + n^2E = 0.\quad (4.16a)$$

Equation (4.16) or (4.16a) shows that the projection of a space line on the image

is a *conic section curve*. And (4.17) shows that the coefficients of the equation may be described *indirectly* in terms of the parameters of the normal $N_S = (l, m, n)$ of plane S . These coefficients actually are related to the elements of the direction vector $V_L = (d_x, d_y, d_z)$ of L by the equality $N_S \cdot V_L = (l, m, n) \cdot (d_x, d_y, d_z) = 0$ because N_S and V_L are perpendicular, or equivalently,

$$ld_x + md_y + nd_z = 0. \quad (4.18)$$

Furthermore, Equation(4.16), as derived above, has a good property that the unknown parameters $l, m,$ and n are confined to appear in just two variables A and B , as shown by (4.17).

When the space line L is vertical, the projection equation described by (4.16a) may be simplified further. Specifically, the direction vector of the vertical line L is $V_L = (d_x, d_y, d_z) = (0, 0, 1)$. Therefore, (16) leads to $0 \times l + 0 \times m + 1 \times n = 0$, or equivalently, $n = 0$. Accordingly, (4.16a) becomes $l^2 u^2 + 2lmuv + m^2 v^2 = 0$, or equivalently, $(lu + mv)^2 = 0$ which describes a line going through the image center at $(0, 0)$ of the form $v = -(l/m)u$. That is, every vertical line in the real-world space becomes a *radial line* in the image space going through the image center, which can be described by

$$v = -Ku, \quad (4.19)$$

where

$$K = \frac{l}{m}. \quad (4.20)$$

Equation 4.19 above has only one parameter K , the slope of the radial line, and this fact facilitates the extraction of the radial line from the image.

4.3 Detection of Vertical Lines in Environments

4.3.1 Initial Detection by Canny Edge Detector

In this section, we introduce the initial step of vertical line detection. After an image is acquired with the camera, the first step is to reduce the resolution of the image. For this, we reduce the image to be $1/4 \times 1/4$ in size to speed up the system operation, and our experimental experience showed that this will not create errors in the detection results. Then, we transform the image in the RGB color model into the YUV model by the following equations:

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B; \\ U &= 0.147R - 0.289G + 0.436B; \\ V &= 0.615R - 0.515G - 0.100B, \end{aligned} \quad (4.21)$$

and use the Y values of the image for the detection work after composing all Y values in the range of 0 to 255 as a gray-level image. Subsequently, we use the canny edge detector to compute edge points in the resulting gray-level image.

In more detail, the function of Canny edge detection can be divided into three steps. The first step is noise reduction. Because the Canny edge detector is easily affected by noise, it uses a filter based on a Gaussian distribution to convolve the raw gray-level image mentioned above. The result looks slightly blurred, compared with the original image. But it can reduce the effect of noise. The second step is to find the intensity gradients of the image. An edge in an image may point to a variety of directions. Therefore, the Canny edge detector uses four filters to detect horizontal, vertical, and diagonal edges. Each edge filter returns a value of the first derivative in the horizontal direction G_x or/and the vertical direction G_y . From this, the edge gradient and direction can be determined by the following equations:

$$G = \sqrt{G_x^2 + G_y^2} ; \quad (4.22)$$

$$\theta = \tan^{-1}\left(\frac{G_y}{G_x}\right). \quad (4.23)$$

Subsequently, the edge direction angle is rounded to one of four angles representing vertical, horizontal and the two diagonals, which are 0, 45, 90 and 135 degrees. The third step is to trace the edges in the image and to conduct hysteresis thresholding. Large intensity gradients are more possible to correspond to edges than small ones. It is impossible to specify a threshold that fit in most cases. Therefore, Canny used thresholding with hysteresis. Thresholding with hysteresis requires two thresholds which are high and low. Making the assumption that important edges should be along continuous curves in the image allows us to follow a small section of the given line and discard noisy pixels that do not constitute a line but have produced large gradients. Therefore, we begin by applying a high threshold. The edges we can be sure are genuine. Starting from the result and using the directional information derived earlier, we can trace the edges in this image. While doing so, we apply the lower threshold as well to trace small sections of the edges as long as we find the starting point. Finally, after this process is completed, we have a binary image where each pixel is marked as either an edge pixel or a non-edge pixel.

4.3.2 Detection of Lines with Widths

In this section, we introduce the method to detect lines with widths. A line-shape object in the real world is not just a line in the mathematical meaning; instead, a line in the image is more like many small line segments lying in the same direction. So we set an area in a line shape with a width as shown as Figure 4.3. After the image is processed by the Canny edge detector algorithm, we use the result to detect lines

with certain widths. The proposed method for such line detection with widths is described as an algorithm in the following. It is noted that the line is continuous in this algorithm, and we will introduce a scheme for detection of broken lines in Section 4.3.3.

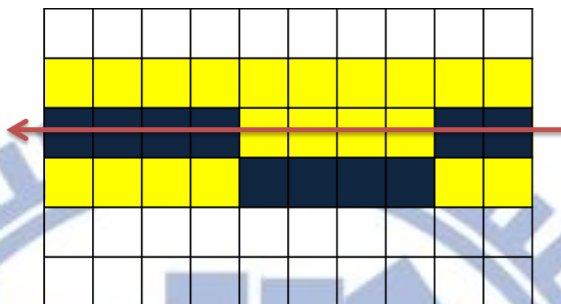


Figure 4.3 An illustration of detection of lines with widths in the image. The black boxes are line points; the yellow area is the region we define as a line; and the red line specifies the direction of this line.

Algorithm 4.1 Detection of Lines with Widths.

Input: a binary image I , a start position of point P_0 , and the pre-set width w and the direction angle θ of the to-be-detected line L .

Output: the length l and the final position P_e of L .

Steps.

- Step 1. Set the position of the scan point P_s to be P_0 .
- Step 2. Set l to be zero.
- Step 3. Compute the set of positions of points $P_n(x_n, y_n)$ which are neighbors of $P_s(x_s, y_s)$ where P_n is on the line L_n which is perpendicular to L and through P_s as shown in Figure 4.4.

3.1 Compute the direction vector $D = (dx, dy)$ of L_n by following equation:

$$\begin{aligned} dx &= \cos(90^\circ - \theta); \\ dy &= \sin(90^\circ - \theta). \end{aligned} \tag{4.24}$$

3.2 Compute the coordinates (x_n, y_n) of all possible P_n by the following equations, where $i = -w/2 \sim w/2$:

$$\begin{aligned} x_n &= i \times dx + x_s; \\ y_n &= i \times dy + y_s. \end{aligned} \quad (4.25)$$

3.3 Compute the Boolean value B by the following equation where b_n is the value of the position P_n in image I :

$$B = b_1 \vee b_2 \vee \dots \vee b_n. \quad (4.26)$$

Step 4. Check whether the Boolean value of B is true or false: if true, set $l = l + 1$ and move P_e to P_s .

Step 5. Move the scan point P_s to the next point by the following equations:

$$\begin{aligned} x_s &= x_s + \cos \theta; \\ y_s &= y_s + \sin \theta. \end{aligned} \quad (4.27)$$

Step 6. Repeat Steps 2 to 5 until P_s is moved to the end of L .

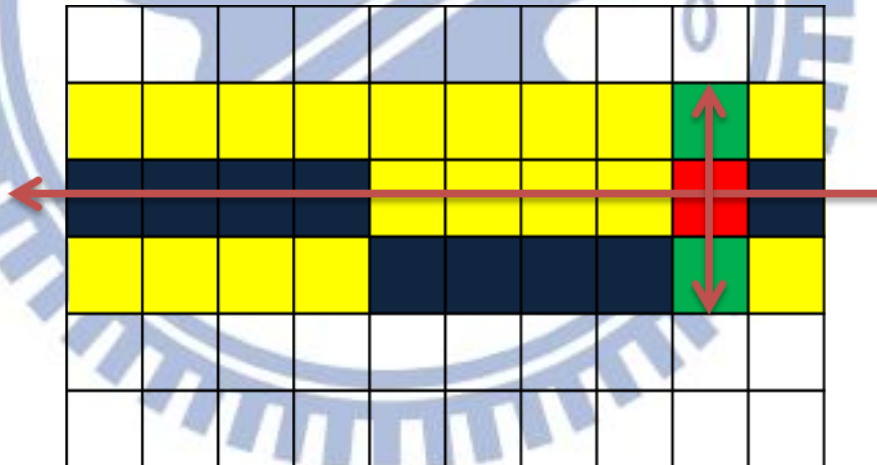


Figure 4.4 An illustration of a line detection. The black painted box is line points. The red painted box is the scan point P_s . The green painted box is the neighbor points P_n .

4.3.3 Detection of Broken Lines

In this section, we introduce the method to detect the broken line. The line in the image acquired by the camera may not be a continuous line when seen with eyes. If the system can detect the broken line, the system will become more accurate and

effective. The idea of detecting the broken line is to calculate the density of the line. A higher density means that it is more possible for a line to be existing. In Figure 4.5(a), we can see a broken line which has a high density of 80%. It must be a line but it is not continuous. It is worth noting that two lines with a gap may be calculated to have a high density as shown in Figure 4.5 (b). The density of it is also 80%, but we can easily recognize them as two lines. So a detection result with a long gap but having a high density is not regarded as a line. The detection of the broken line is described as an algorithm in the following.

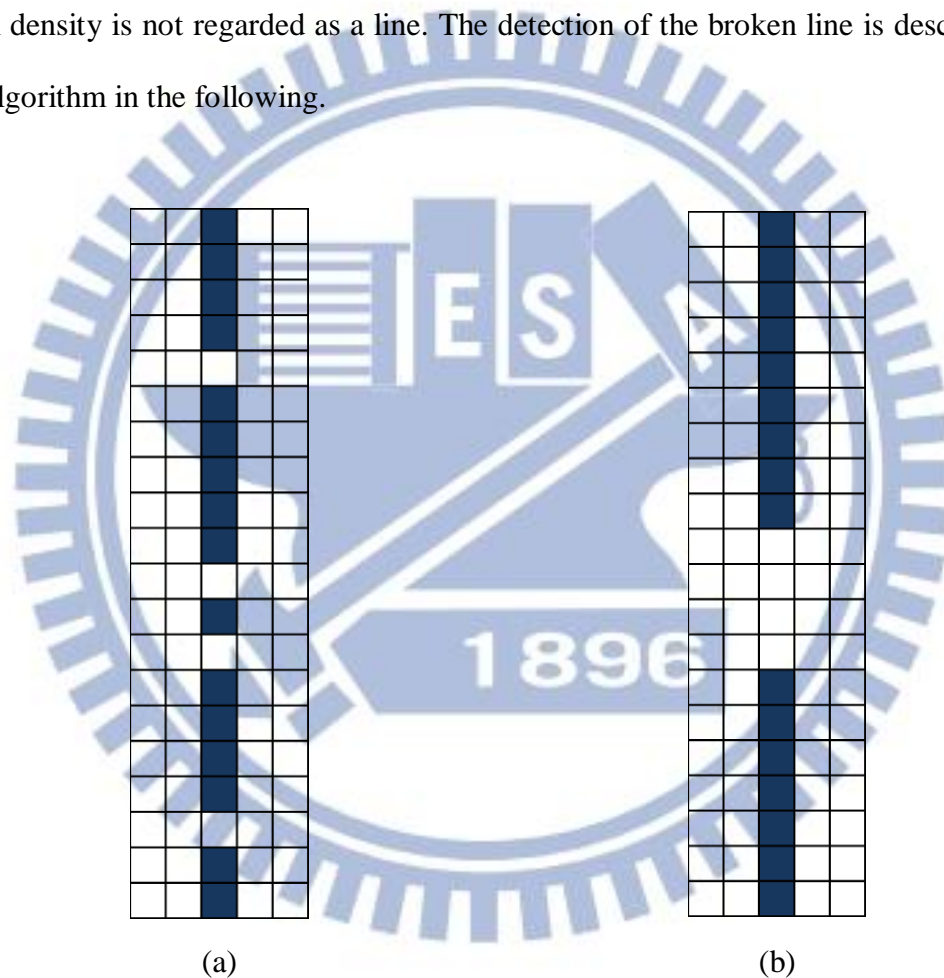


Figure 4.5 An illustration of a broken line detection. (a) A broken line with a 80% density. (b) Two lines whose overall density is 80%.

Algorithm 4.2 Detection of Broken Lines.

Input: a binary image I , a start position of point P_0 , a threshold g of the gap size, and the direction angle θ of the line L .

Output: a set S including the length l and the density d of the detected L .

Steps.

- Step 1. Set the position of scan point P_s to be P_0 .
- Step 2. Move P_s to the next point in the direction of L by using Equation 4.27.
- Step 3. Repeat Step 2 until the Boolean value of the position of P_s is true in I .
- Step 4. Set the value of l to be zero, set the value of accumulating line points acc to be zero, and set a value g_c to be g to count the number of gap points.
- Step 5. Move P_s to the next point in the direction of L by using Equation 4.27.
- Step 6. Set $l = l + 1$.
- Step 7. Set $acc = acc + 1$ if P_s is true.
- Step 8. If P_s is false, set $g_c = g_c - 1$; else, set g_c to be g .
- Step 9. Repeat Steps 4 to 7 until the value of g_c is equal to zero.
- Step 10. Save the value of l into S .
- Step 11. Compute d by the following equation and save it into S :
$$d = \frac{acc}{l}. \quad (4.28)$$
- Step 12. Repeat Steps 2 to 10 until P_s is moved to the end of L .

4.4 Algorithm for Vertical Line Detection

In this section, we introduce the algorithm for vertical line detection. The algorithm includes all the schemes described before to detect vertical lines in omni-images. First, the vertical line in the real-world space becomes a radial line in the omni-image as illustrated in Figure 4.6. Next, we set the center of the image as an

area, shown as the red points in Figure 4.6. The vertical lines in the real world space may not be vertical completely. The system uses these points as the center of a circle to find the radial lines by searching lines of all directions like a radar. The use of an area as the circle center can endure to some degree the phenomenon of incomplete verticalness just mentioned, and improve the effectiveness of the proposed method, which is The described as an algorithm in the following.

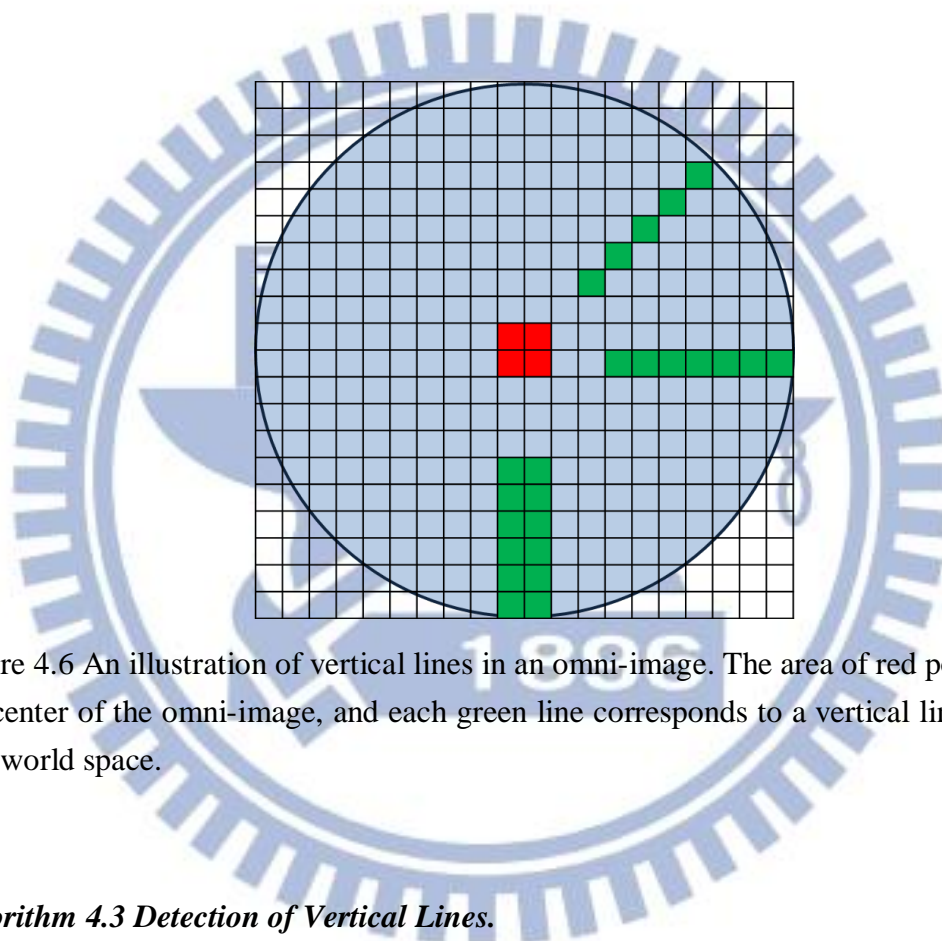


Figure 4.6 An illustration of vertical lines in an omni-image. The area of red points are the center of the omni-image, and each green line corresponds to a vertical line in the real-world space.

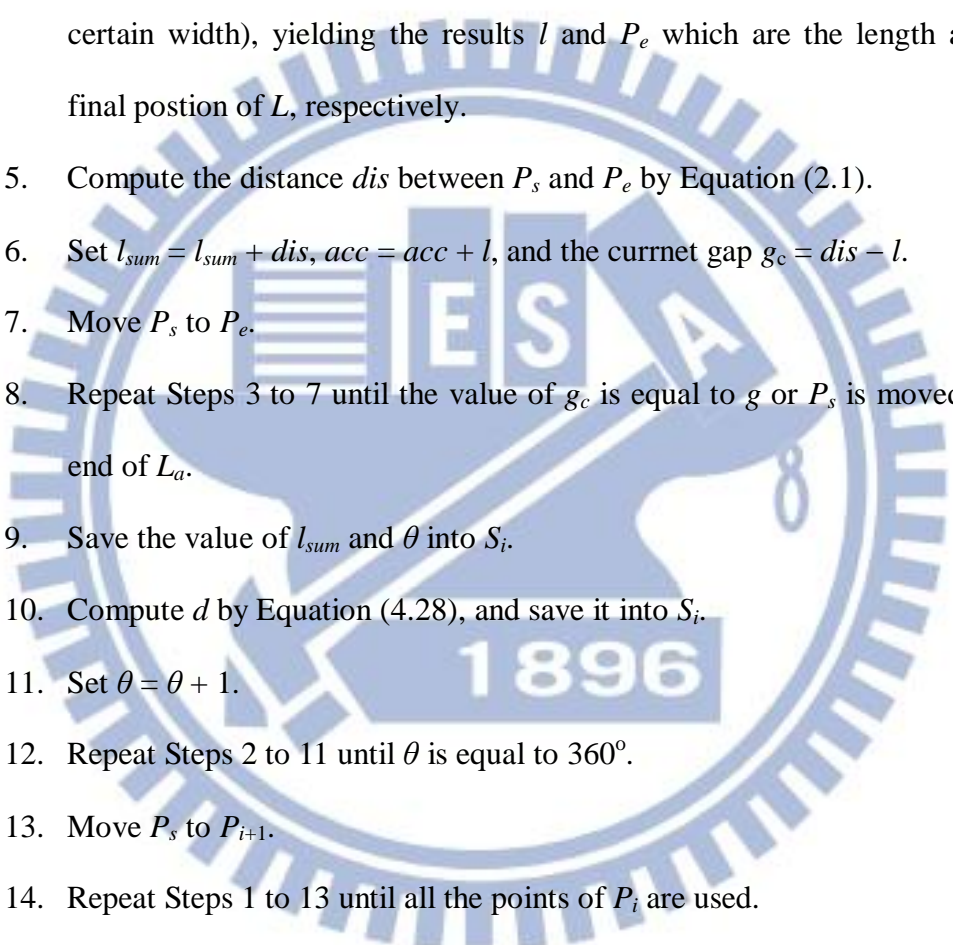
Algorithm 4.3 Detection of Vertical Lines.

Input: a binary image I , a set of positions of center points P_i , a threshold g of gap size, and the width w of the vertical line to be detected.

Output: a set S_i for each detected line L which includes the length l_{sum} , the density d , and the direction angle of the line.

Steps.

Step 1. Set the position of the scan point P_s to be P_i .

- 
- Step 2. Set the direction angle θ of the to-be-detected line L_a to be zero.
- Step 3. Set l_{sum} to be zero; define acc as the value of accumulated line points with its initial value set to be zero as well; and set g_c as a value to count the number of gap points with g as its maximum value.
- Step 4. With the image I , the width w , the angle θ and the point P_s as inputs, perform Algorithm 4.1 to conduct detection of thick lines (i.e., with a certain width), yielding the results l and P_e which are the length and the final position of L , respectively.
- Step 5. Compute the distance dis between P_s and P_e by Equation (2.1).
- Step 6. Set $l_{sum} = l_{sum} + dis$, $acc = acc + l$, and the current gap $g_c = dis - l$.
- Step 7. Move P_s to P_e .
- Step 8. Repeat Steps 3 to 7 until the value of g_c is equal to g or P_s is moved to the end of L_a .
- Step 9. Save the value of l_{sum} and θ into S_i .
- Step 10. Compute d by Equation (4.28), and save it into S_i .
- Step 11. Set $\theta = \theta + 1$.
- Step 12. Repeat Steps 2 to 11 until θ is equal to 360° .
- Step 13. Move P_s to P_{i+1} .
- Step 14. Repeat Steps 1 to 13 until all the points of P_i are used.

4.5 Experimental Results

We have described the proposed methods for detecting vertical-line features around the vehicle in omni-images before. In the first part, we described the idea of detection of vertical lines in omni-images. Figure 4.7 shows the phenomenon that the

vertical line in the real-world space becomes a radial line in the omni-image. In addition, the canny edge detector is applied as the initial process of line detection. Figure 4.8 shows the result of Canny edge detection applied to Figure 4.7. And Figure 4.9 shows the result of vertical line detection using the proposed method. The red line shows the line detected by the system.

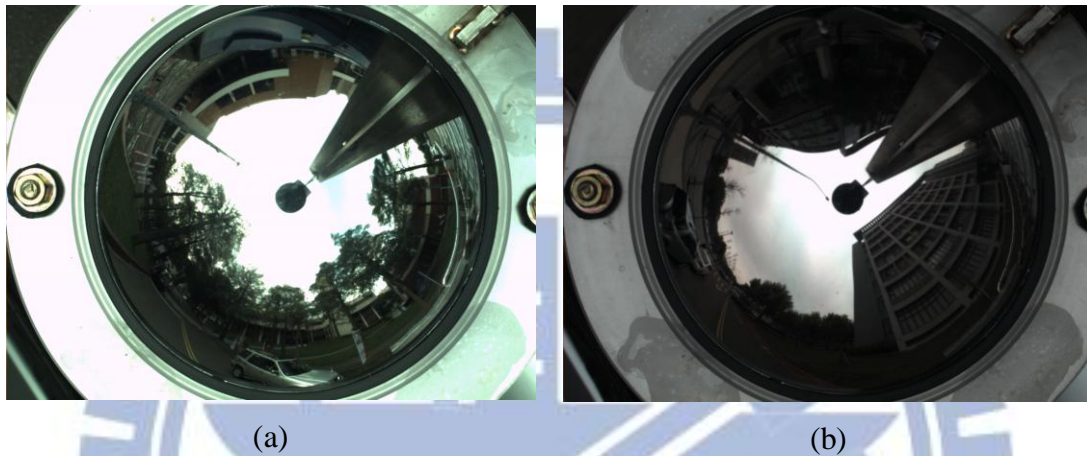


Figure 4.7 Illustrations of the phenomenon that a vertical line become a radial line in the omni-image. (a) Scene 1. (b) Scene 2.

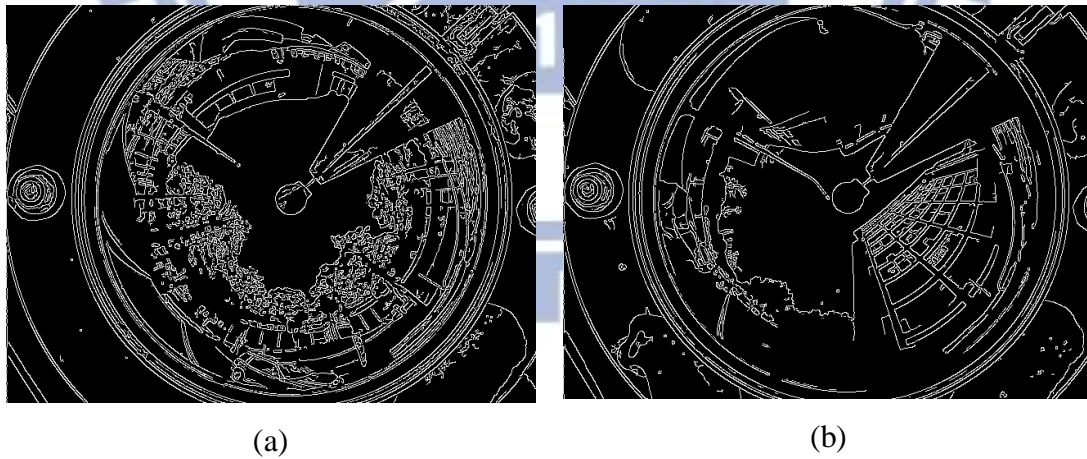
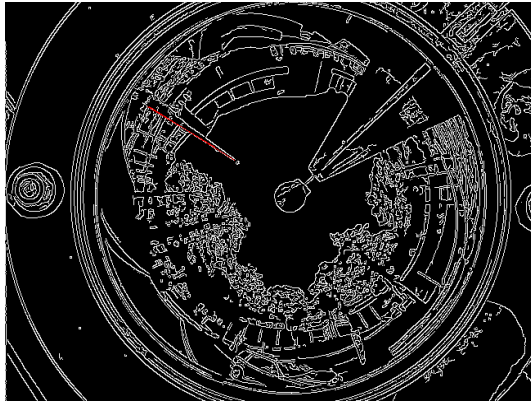
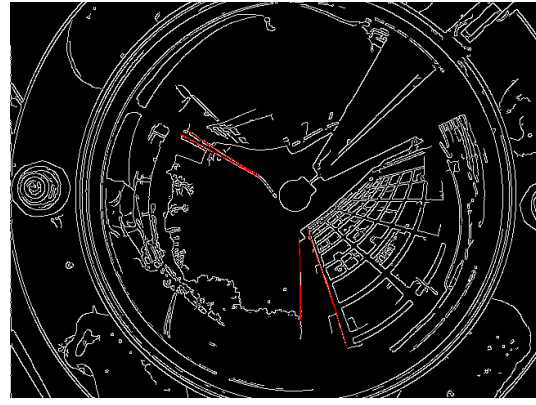


Figure 4.8 Results of Canny edge detection. (A) Result of Figure 4.7(a). (B) Result of Figure 4.7(b).

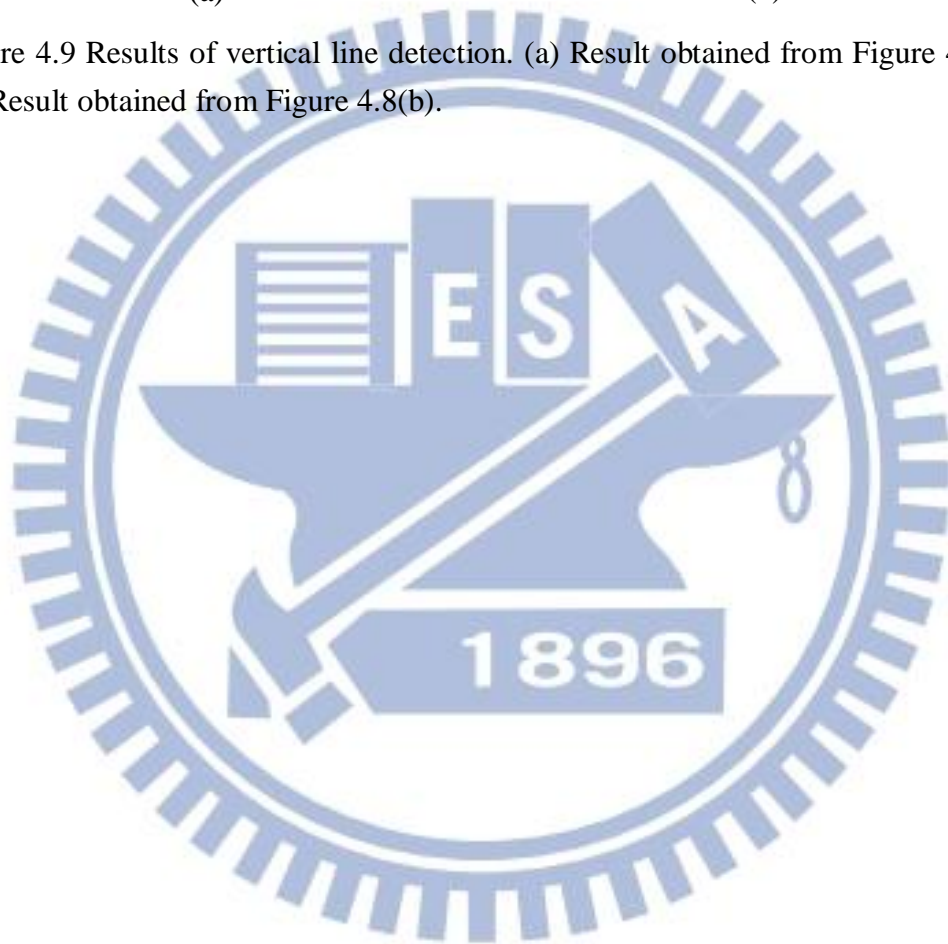


(a)



(b)

Figure 4.9 Results of vertical line detection. (a) Result obtained from Figure 4.8(a).
(b) Result obtained from Figure 4.8(b).



Chapter 5

Vehicle Localization for Tour Guidance in Outdoor Park Areas by Computer Vision Techniques

5.1 Introduction

In this chapter, we introduce the proposed method for vehicle localization for navigation in outdoor park areas by computer vision techniques. The vehicle localization method is needed for the system to navigate and compute building positions for AR display. The first part of the vehicle localization work is to estimate the vehicle speed.

In more detail, the speed is estimated in this study by computing the motion vectors of the vehicle in acquired images. It is impossible that there exist features to detect at every spot on the path. In other words, the system must have a method to locate the vehicle without using features at certain spots on the path. In this study, we use the vehicle speed to solve this problem. The detail is described in Section 5.2.

On the contrary, in normal cases we use the correlation between vertical lines in the environment, when available, and the vehicle to locate the vehicle. Specifically, the system detects a single feature to locate the vehicle in certain cases; and in other cases, the system uses multiple features to locate the vehicle. The more features the system detects, the more accurately the system conducts the vehicle localization task.

Moreover, we use the longest common subsequence (LCS) algorithm to

implement the system's work of vehicle localization using multiple features. The detail is described in Section 5.4. Finally, we analyze the information about the environment to make the system to detect vertical lines more accurately. The detail is described in Section 5.5.

5.2 Estimation of Vehicle Speed

In this section, we introduce the proposed method to estimate the vehicle speed. At certain positions, the system cannot use features to locate the vehicle due to unavailability of appropriate local features. So we propose a method to use the result of motion-vector estimation to compute the vehicle speed, and use the estimated speed to locate the vehicle. The detail is described in the following.

5.2.1 Computation of Motion Vectors

To estimate the vehicle speed, we compute motion vectors using acquired images. An illustration of motion estimation for each macroblock in an image is depicted in Figure 5.1. For each macroblock B in the current image frame, a search window in the reference frame is searched for the best-match macroblock B' with respect to B . The computation of motion vectors is described as an algorithm in the following.

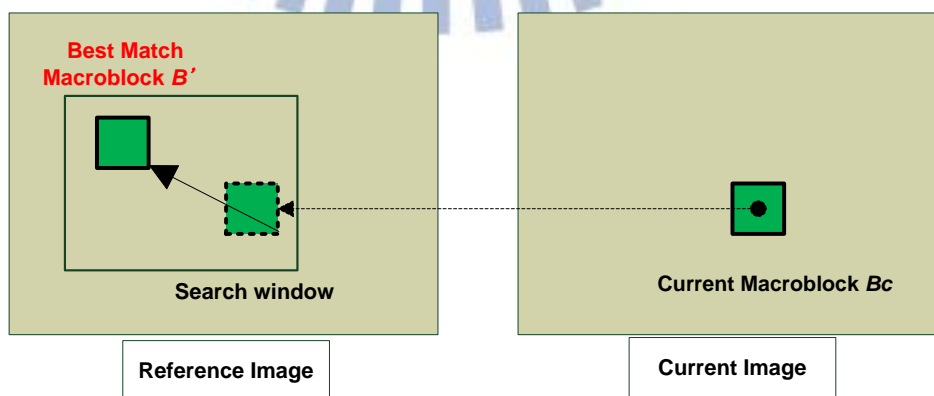


Figure 5.1 An illustration of searching for the best-match macroblock.

Algorithm 5.1 Computation of Motion Vectors.

Input: a reference image I_r , the current image I_c , and a $k \times l$ search window W .

Output: a set S of motion vectors.

Steps.

Step 1. Divide the current image I_c into macroblocks B_0 through B_n of size 16×16 .

Step 2. Set the current block B_c in the current image I_c to be B_k with $k = 0$.

Step 3. Search the best-match macroblock in the search window W within the reference image I_r by the following steps.

3.1 Set the search block B_s to be at the position (i_0, j_0) of the left-upper side with the distances from B_s to B_c being k and l , respectively, in the two axis directions in the image coordinate system.

3.2 Compute the cost function $Cost_k$ by the following formula where $C(a, b)$ is the value of the pixel at coordinates (a, b) in I_c , and $R(a, b)$ is the value of the pixel at coordinates (a, b) in I_r , and the coordinates of B_s are (i_k, j_k) :

$$Cost_k = \frac{1}{16^2} \sum_{k=0}^{15} \sum_{l=0}^{15} |C(x+k, y+l) - R(x+i+k, y+j+l)|. \quad (5.1)$$

3.3 Move B_s to the next position by shifting a pixel in a raster scan order.

3.4 Save the motion vector $V_s(v_x, v_y)$ computed in the following way if

$Cost_k$ is smaller than every other $Cost_j$:

$$\begin{aligned} v_{kx} &= i_k - i_0; \\ v_{ky} &= j_k - j_0. \end{aligned} \quad (5.2)$$

3.5 Repeat Steps 3.1 to 3.4 until all possible blocks in the window are processed.

Step 4. Move B_c to the next block.

Step 5. Repeat Steps 3 and 4 until all the blocks are searched.

At the end of executing the above algorithm, each block B_k in the current image I_c will have a corresponding motion vector $V_k(v_{kx}, v_{ky})$.

5.2.2 Vehicle Speed Estimation Using Motion Vectors

In this section, we describe a method we propose to estimate the vehicle speed using motion vectors by modifying the algorithm which is described in Section 5.2.1. The resulting algorithm will make our system more accurate. In more detail, the motion vectors we compute include many directions. But the vehicle is driven just forward in one direction. So, the valid motion vectors must all be in the same direction; motion vectors in the other directions might be noise or errors due to incorrect detection results. Furthermore, not all the motion vectors in the entire omni-image are useful for our purpose of vehicle localization. Therefore, we cut the part of the omni-image which only includes the road for our use, as shown in Figure 5.2. The motion vectors in this part should be in the same direction. So, we use this image part to compute motion vectors. Finally, we average all the motion vectors in the vehicle direction, and compute the vehicle speed accordingly. An algorithm describing the above process is presented in the following.

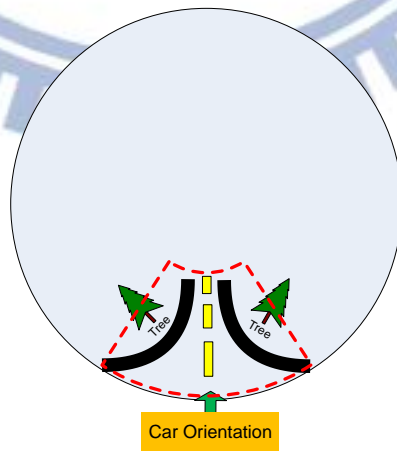


Figure 5.2 An illustration of cutting a part of the omni-image for motion vector computation. The area enclosed by the red line is the part we cut.

Algorithm 5.2 Estimating the vehicle speed.

Input: a reference omni-image I_r , the current omni-image I_c , a search window W with size $k \times l$, the direction $D(d_x, d_y)$ of the vehicle, and two threshold values T_u and T_d .

Output: a value s of the vehicle speed.

Steps.

- Step 1. Cut out parts of I_r and I_c as I_r' and I_c' , respectively, in which the road appears, as shown in Figure 5.2.
- Step 2. Use the cut image I_r' and I_c' and the $k \times l$ search window W as inputs, perform Algorithm 5.1 to compute motion vectors, and save the result into a set S .
- Step 3. Delete those motion vectors $V_i(v_x, v_y)$ of S that are not in the direction of D by the following steps.
 - 3.1 If one of d_x and v_x is positive and the other is negative, then delete V_i .
 - 3.2 If one of d_y and v_y is positive and the other is negative, then delete V_i .
- Step 4. Delete v_i if v_i is larger than T_u or smaller than T_d .
- Step 5. Compute the mean value of v_i by the following equation and take the result as the desired vehicle speed s , where N is the number of the remaining motion vectors:

$$s = \frac{\sum v_i}{N}. \quad (5.3)$$

5.3 Vehicle Localization by Single Line Features

Vehicle localization is necessary for the system to navigate in a tour. By vehicle localization, the positions of buildings can be computed and used to show the AR

information about the building. In this study, we propose to locate the vehicle by using line features. In more detail, the features in omni-images are detected by Algorithm 4.3. Then, we use the angles of the resulting features and the learned data to locate the vehicle. In the simple case, the system uses only one feature to locate the vehicle. The detail is described in the following.

5.3.1 Idea of Vehicle Localization by Line Features

After detecting the vertical lines in an omni-image, the system uses the resulting angle of the line feature in the image to locate the vehicle as shown in Figure 3.4. Because the feature line is learned into the environment map, we can use the relation between the feature and the vehicle on the path to locate the vehicle position. And the relation between them may be expressed in terms of angle. The vehicle in different positions will yield features of different angles, as can be figured out from Figure 5.3, one angle of the feature corresponds to one position in the path uniquely. Accordingly, we can use the detected line feature to calculate the vehicle position on the path by Equation 3.3. And the algorithm is shown in Section 5.3.2.

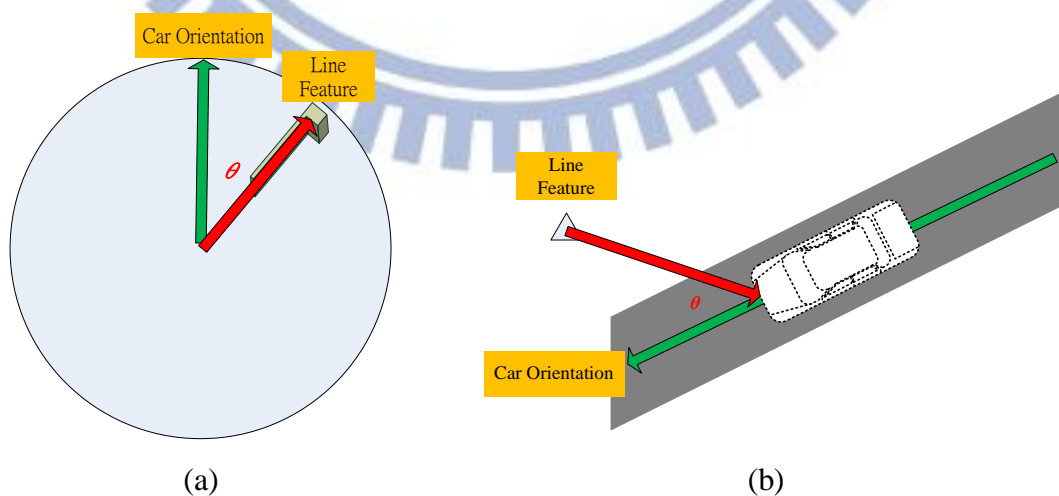


Figure 5.3 An illustration of locating the vehicle. (a) An omni-image with a detected line feature. (b) An illustration of locating the vehicle on the map.

5.3.2 Algorithm for Vehicle Localization by Single Line Features

In this study, we propose an algorithm to locate the vehicle by single line features. The environment map is one of the inputs of this algorithm, which includes the learned data of the path and the detected vertical-line feature. Moreover, the angle of the direction of the line which the system detected is also the input to the algorithm.

Algorithm 5.3 Vehicle Localization by Single Line Features.

Input: the angle θ_d of a detected line L , the feature F which corresponds to the line L , and a path segment p_s .

Output: the vehicle position $P_v(x_v, y_v)$.

Steps.

- Step 1. Check the angle θ_d whether is in the range from the *first-angle* θ_f to the *last-angle* θ_l which are learned in learning stage in chapter 3. If it is not in the range, do not use the this feature to locate the vehicle.
- Step 2. Get the angle θ_p of the path segment p_s from the learned data of the path in the map Map .
- Step 3. Get the position $P_f(x_f, y_f)$ of the feature F from the learned feature data in the map Map .
- Step 4. Compute the angle θ_f of the direction from the feature to the vehicle by the following equation:

$$\theta_f = -(\theta_p + \theta_d) \quad (5.4)$$

where θ_d is the angle of the detected line L .

- Step 5. Compute the line $L(x_l, y_l)$ which goes through two points P_f and P_v by the

following equation:

$$\begin{aligned}x_l &= x_f + \cos(\theta_f) \times t; \\y_l &= y_f + \sin(\theta_f) \times t.\end{aligned}\tag{5.5}$$

Step 6. Compute the vehicle position $P_v(x_v, y_v)$ by the following steps:

- 6.1 Get the line equation $F(x, y)$ of the path segment p_s from the map *Map*.
- 6.2 Find the intersection point of $F(x, y)$ and $L(x_l, y_l)$ by substituting the equations of L described by (5.5) into $F(x, y)$ and solving the result to get t .
- 6.3 Substituting t into Equation 5.5, and the solution (x_l, y_l) specifies the desired $P_v(x_v, y_v)$.

5.4 Vehicle Localization by Multiple Line Features

In this section, we introduce the proposed method to locate the vehicle by multiple line features. In some cases, the system can use multiple features to locate the vehicle. The more features the system detects, the more accurately the system conducts vehicle localization. But we need a method to match the features seen in the current image against those in the map. So we propose a method which can conduct such matching, which is based on the longest common subsequence (LCS) algorithm.

5.4.1 Review of Longest Common Subsequence (LCS) Algorithm

The longest common subsequence (LCS) algorithm aims to find the longest subsequence common to all sequences in a set of sequences. Note that a subsequence

is a sequence that can be derived from another sequence by deleting some elements without changing the order of the remaining elements. In this study, we use the angles of feature lines as inputs to this algorithm, i.e., we regard such angles as the elements of sequences and match the elements by the LCS algorithm. But we have to modify the equality conditions used in the traditional LCS algorithm. because the angles detected by the system may have errors. Specifically, we set a tolerant range for defining whether two angles are equal or not. The algorithm is shown in the following.

Algorithm 5.4 Longest common subsequence.

Input: two angle sequences $S_x = \langle x_1, x_2, \dots, x_n \rangle$ and $S_y = \langle y_1, y_2, \dots, y_m \rangle$ where all x_i and y_j are angles of vertical lines; and a threshold d for judging the equality of two angles.

Output: the length L of the longest common subsequence.

Steps.

- Step 1. If $n = 0$ or $m = 0$ set L to be zero.
- Step 2. If the absolute value of $x_n - y_m$ is smaller than d , then do the following steps.
 - 2.1 With $S'_x = \langle x_1, x_2, \dots, x_{n-1} \rangle$, $S'_y = \langle y_1, y_2, \dots, y_{m-1} \rangle$ and d as inputs, perform Algorithm 5.4 to compute the length L' of the LCS of S'_x and S'_y .
 - 2.2 Set $L = L' + 1$.
- Step 3. If the absolute value of $x_n - y_m$ is larger than d , then do the following steps.
 - 3.1 With the S_x , S_y , and d as inputs, perform Algorithm 5.4 to compute the length L_n of the LCS of S_x and S'_y .

- 3.2 With the S_x' , S_y , and d as inputs, perform Algorithm 5.4 to compute the length L_m of the LCS of S_x' and S_y .
- 3.3 If L_n is larger than L_m , set L to be L_n ; else, set L to be L_m .

5.4.2 Vehicle Localization using Multiple Features by LCS Algorithm

In this study, we use vertical line as a new type of feature to locate the vehicle. In other words, the position where the system can detect many features is a different case for locating the vehicle. In Chapter 3, we introduce a method to learn multiple line features. And we use the learned data, which include every position where the vehicle can “see” features, to locate the vehicle on the path. Then, the system just matches the angles of those “seen” features to find the best match position. But a problem arises there in matching the features — because the features we detect are not recognized to be correct, the detected line might be really a feature or just noise. And the features we learned might not be detected by the system while the vehicle is driven on the path. The LCS algorithm is modified to solve this problem in this study. The system uses this algorithm to match those angles of vertical lines even if not all the features around the vehicle are completely detected. These ideas of proposed vehicle localization using multiple vertical-line features are described as an algorithm in the following.

Algorithm 5.5 Vehicle localization by multiple features.

Input: an angle sequence $S_x = \langle x_1, x_2, \dots, x_n \rangle$ detected by the system; a threshold d for judging the equality of two angles; and a set S_e of learned data which include the vehicle position P_i and an angle sequence S_i .

Output: the vehicle position P_v and the length L of the subsequence computed by Algorithm 5.4.

Steps.

- Step 1. Set L to be zero, and i to be zero as well.
- Step 2. With S_i , S_x and d as inputs, perform Algorithm 5.4 to compute the length L' of the LCS of S_i and S_x .
- Step 3. If L' is larger than L , set L to be L' , and set P_v to be P_i which is included in the learned data and corresponds to S_i .
- Step 4. Set $i = i + 1$.
- Step 5. Repeat Steps 2 to 4 until i equals the number of S_e .

5.5 Knowledge-based Analysis of Tours

In this section, we introduce a method we propose in this study by knowledge-based analysis to make the system more accurate in vehicle localization. The vertical lines are not always detected or the detection result sometimes will be just noise. So, the proposed method tries to filter the noise and make the system more accurate.

5.5.1 Uses of Knowledge about Environments

The main idea of the proposed method is to detect the right feature at the right position. Specifically, the system uses the image sequence to analyze the position of the vehicle. Firstly, the omni-image we use to detect features can divide into two parts, the left side and the right one, as shown in Figure 5.4(a). The omni-camera can “see” all the environment around the vehicle in 360 degrees. And the feature lines can be

seen in the left side of the road or in the right side. So we can define the features in the two sides when learning the feature. The two sides of features would be detected individually. Secondly, the vehicle is driven just forward. So the detected features must move backward in the image sequence as shown in Figure 5.4(b). Moreover, the system predicts the position of the feature and detects it.

In more detail, in the proposed method we calculate the vehicle speed and then use the last position of the vehicle to predict the next position of the vehicle on the environment map. Next, we use the predicted position of the vehicle on the environment map to calculate the predicted position of the features. Then, we use such information to find out the feature in the detection area in the image as depicted in Figure 5.4. If there is no feature detected in the area, we predict that no feature is detected by the system. The system would not regard the feature as wrong one. Finally, in the learning of features, we have recorded the positions where the feature can be seen. So the system can check the position of the vehicle and the feature to see if the position fits the learning data.

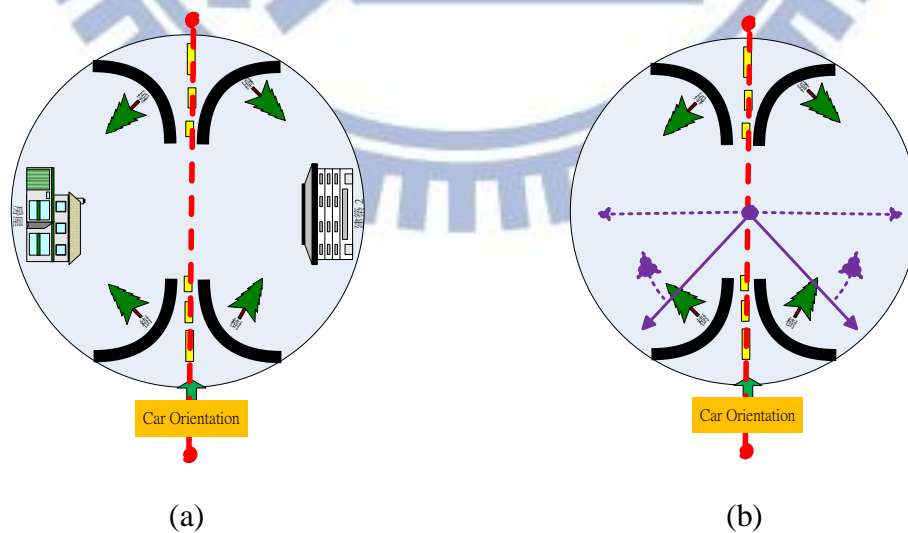


Figure 5.4 An Illustration Of Omni-Image. (A) The Omni-Image Which Divided Two Parts. (B) Two parts of image and the purple line is direction of system to detect the line features.

5.5.2 Algorithm for Vehicle Localization in Tours

The algorithm for vehicle localization includes all the methods to locate the vehicle described previously. In the general case, the system uses a single feature to locate the vehicle. In some areas, the system can use multiple features to locate the vehicle. And at certain positions, the system cannot use features to locate the vehicle due to unavailability of appropriate local features, and uses the vehicle speed to locate the vehicle.

Algorithm 5.6 Vehicle localization.

Input: an environment map Map which includes the learned data, the last position $P_l(x_l, y_l)$ of the vehicle, a reference omni-image I_r , the current omni-image I_c , a search window W with size $k \times l$, the direction $D(d_x, d_y)$ of the vehicle, two threshold values T_u and T_d , an angle sequence $S_x = \langle x_1, x_2, \dots, x_n \rangle$ detected by the system, and a threshold d for judging the equality of two angles.

Output: the vehicle position P_v .

Steps.

Step 1. Predict the vehicle position $P(x_p, y_p)$ using the vehicle speed by the following steps.

1.1 With $I_r, I_c, W, D(d_x, d_y)$ and the two threshold values T_u and T_d as inputs, perform Algorithm 5.2 to compute the speed s of the vehicle.

1.2 Use the vehicle speed to compute the vehicle position by the following equation where θ is the angle of the path obtained from Map :

$$\begin{aligned} x_p &= x_l + \cos(\theta) \times s; \\ y_p &= y_l + \sin(\theta) \times s. \end{aligned} \tag{5.6}$$

Step 2. Use the predict position P to find the set S_f of features around P on Map .

- Step 3. If the set S_f is empty, use the vehicle speed to locate the vehicle, and set P_v to be P .
- Step 4. If only one feature F is in S_f , use the single feature to locate vehicle by the following steps.
- 4.1 Find the angle θ which is the closest angle of the direction of P with respect to F .
 - 4.2 With F , the path segment p_s from Map and θ as inputs, perform Algorithm 5.3 to compute the position of vehicle.
- Step 5. If many features F_i are in S_f , use the multiple features to locate vehicle in the following way: with $S_x = \langle x_1, x_2, \dots, x_n \rangle$, a threshold d and a set S_f as inputs, perform Algorithm 5.5 to compute the position of the vehicle.

5.6 Experimental Results

We have described the proposed methods for locating the vehicle. As described in Section 5.2 we used motion vectors to compute the vehicle speed. An example of detected motion vectors is shown in Figure 5.5. As described in Section 5.3, we also used single features to locate the vehicle. Figures 5.6(a) and 5.6(b) show an example of detecting just one feature to locate the vehicle. Finally, as described in Section 5.4, we used multiple features to locate the vehicle, as shown by the example in Figure 5.7.

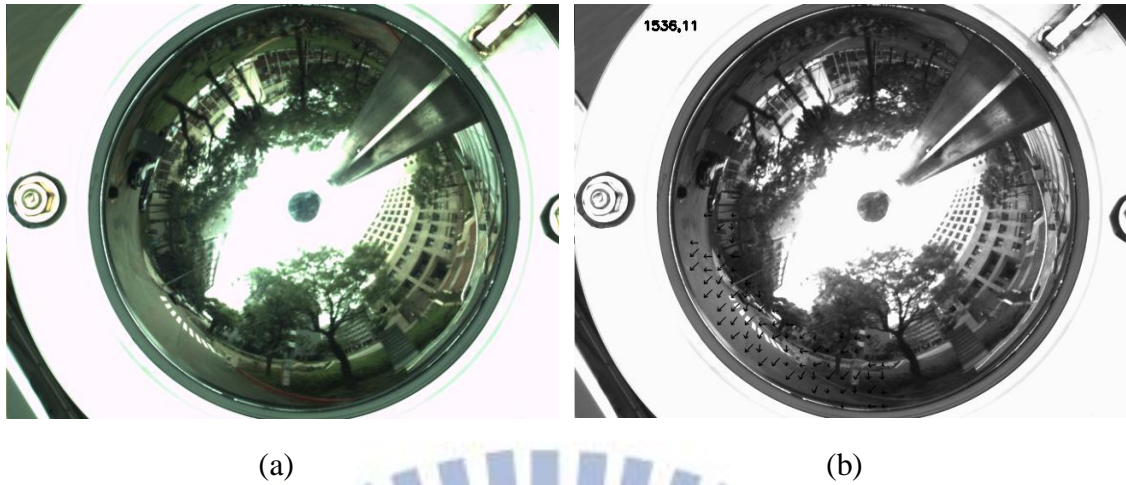


Figure 5.5 Detected motion vectors in an omni-image. (a) The original image. (b) The detected motion vectors.

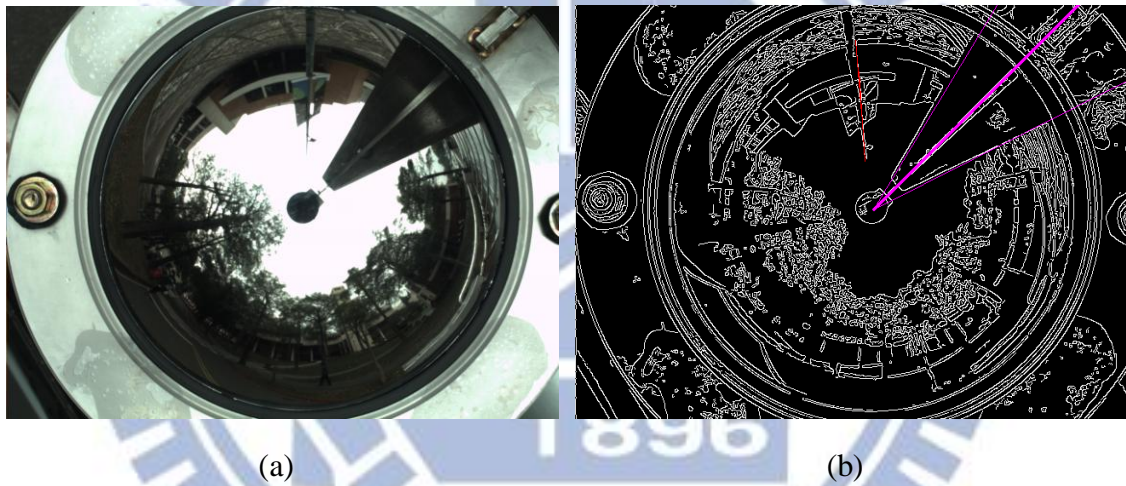
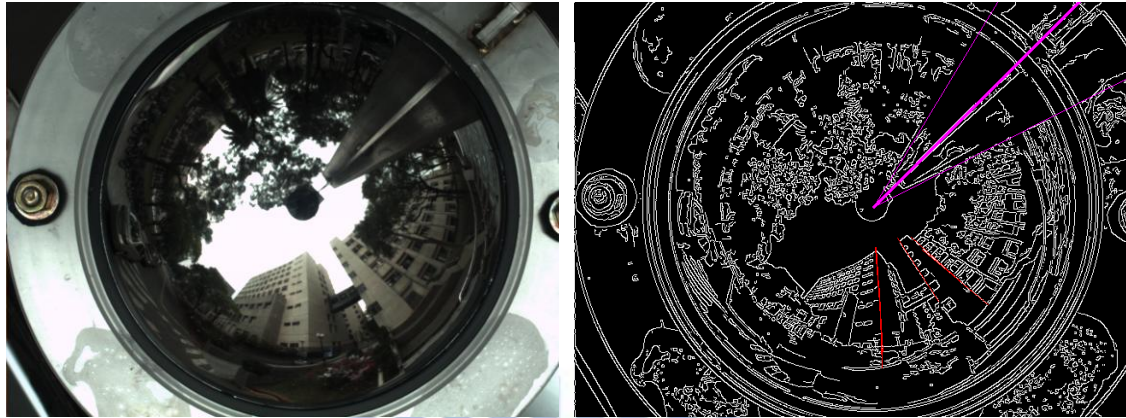


Figure 5.6 The localization of vehicle by using single feature. (a) The omni-image acquired from the camera. (b) A binary omni-image, in which the red line is the detected line feature. (c) The map showing the position of vehicle where the red point is the position of the vehicle and the blue points are the positions of features.



(a)

(b)



(c)

Figure 5.7 Vehicle localization by using multiple features. (a) The omni-image acquired from the camera. (b) A binary omni-image, in which the red lines indicate detected line features. (c) The map showing the position of vehicle, where the red point is the position of the vehicle and the blue points are the positions of the detected features.

Chapter 6

Proposed Augmented Reality-Based Tour Guidance Using an Omni-camera

6.1 Ideas of Proposed Techniques

In this study, we use the AR technique to show the information about the environment, especially about the along-path buildings, while driving the vehicle in there. To show the AR information, we need the positions of the buildings. Therefore, we use the location of the vehicle and the learned data to compute the positions of the buildings.

In more detail, at first the system detects the line features which can be “seen” along the path while the vehicle is being driven. Next, the system uses the detected features to locate the position of the vehicle. Furthermore, by using the learned data and the location of the vehicle, the system can compute the relative position between the vehicle and the building. Then, the system computes the position of the building on the mobile device screen to show the AR information. Note that the camera used in this study is an omni-camera. Because the image acquired by this camera is distorted, we transform the omni-image into perspective view images as well before let it be displayed on the mobile device screen, so that the user can see this image intuitively. In the meantime, the system can also take the advantage of larger perspective views to detect features.

6.2 Construction of Images from Front Passenger's View

In this section, we describe the details of the method proposed by Jeng and Tsai [4] to generate perspective-view images from omni-images acquired with the omni-imaging device used in this study. Their method is based on the use of a so-called space-mapping table, which maps each image point to a space point without involving the camera intrinsic and extrinsic parameters.

6.2.1 Construction of Image-to-space Mapping Table

To create the space-mapping table before it can be used for coordinate mapping, it is essential to select some world space points with known positions and the corresponding points in the omni-image. It is known that a point p in the image space is formed by all the world space points which lie on an incoming light ray R , as illustrated in Figure 6.1.

More specifically, at first three assumptions are made: (1) O_m is the focal point of the hyperboloidal-shaped mirror in the omni-camera; (2) O_w is on the mirror bottom plane of the mirror; and (3) P_1 and P_2 are two world-space points on the light ray R . In addition, suppose that the image point corresponding to both P_1 and P_2 is denoted by p . Then, we have the corresponding point pairs (P_1, p) and (P_2, p) which can be used to generate the pan-mapping table. However, if we took erroneously O_w instead of O_m as the focal point, then P_1 and P_2 would lie on different light rays, though the corresponding image points are still a single one, namely, p . In this way, an incorrect space-mapping table will be generated. To avoid this error, we must find out the real position of the focal point of the hyperboloidal-shaped mirror.

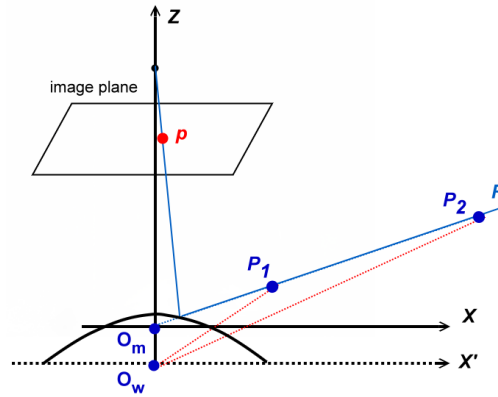


Figure 6.1 The space points and their corresponding image points.

For this purpose, as shown in Figure 6.2 we use two different landmark points L_1 and L_2 with known heights and horizontal distances from the axis of the mirror. Both of L_1 and L_2 are projected onto the same image point p . Then, according to the geometry shown in Figure 6.2, the position of the focal point O_m with respect to the center point O_w of the bottom plane of the mirror may be computed by the following equations:

$$\tan \theta = \frac{H_1 - \overline{O_m O_w}}{D_1} = \frac{H_2 - H_1}{D_2 - D_1};$$

$$\overline{O_m O_w} = H_1 - D_1 \times \tan \theta = H_1 - D_1 \times \frac{H_2 - H_1}{D_2 - D_1}. \quad (3.1)$$

Moreover, we describe here the method proposed by Jeng and Tsai [4] to build a pano-mapping table using the coordinate data of the landmark point pairs. The two-dimensional pano-mapping table with the horizontal and vertical axes being the azimuth angle θ and the elevation angle ρ , respectively, is illustrated in Table 6.1, which is described next.

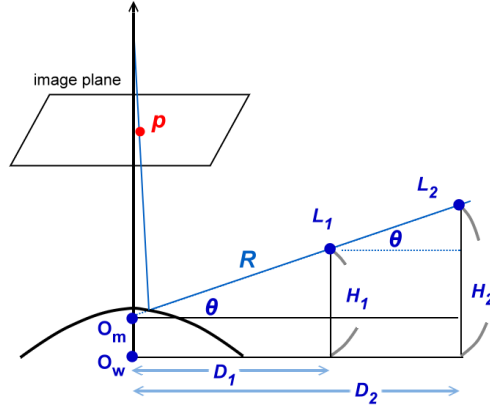


Figure 6.2 Finding out the focal point O_m .

Table 6.1 Example of pano-mapping table of size $M \times N$

	θ_1	θ_2	θ_3	θ_4	...	θ_M
ρ_1	(u_{11}, v_{11})	(u_{21}, v_{21})	(u_{31}, v_{31})	(u_{41}, v_{41})	...	(u_{M1}, v_{M1})
ρ_2	(u_{12}, v_{12})	(u_{22}, v_{22})	(u_{32}, v_{32})	(u_{42}, v_{42})	...	(u_{M2}, v_{M2})
ρ_3	(u_{13}, v_{13})	(u_{23}, v_{23})	(u_{33}, v_{33})	(u_{43}, v_{43})	...	(u_{M3}, v_{M3})
ρ_4	(u_{14}, v_{14})	(u_{24}, v_{24})	(u_{34}, v_{34})	(u_{44}, v_{44})	...	(u_{M4}, v_{M4})
...
ρ_N	(u_{1N}, v_{1N})	(u_{2N}, v_{2N})	(u_{3N}, v_{3N})	(u_{4N}, v_{4N})	...	(u_{MN}, v_{MN})

The procedure for constructing the pano-mapping table includes three major stages: (1) landmark learning, (2) estimation of the coefficients of a radial stretching function describing the geometry of the mirror reflection in the omni-camera, and (3) pano-mapping table creation.

1. Landmark learning

Landmark learning is a procedure in which several pairs of world space points with known positions and their corresponding pixels in a taken omni-image are selected for constructing the pano-mapping table. The omni-camera is first set horizontally on the ground with both its mirror base plane and omni-image plane

parallel to the ground, which is just the X - Y plane of the WCS. Then, a sufficient number (more than six) of points in the world space, called *landmark points* hereafter, are selected, and the coordinates of them are measured *manually* with respect to the previously-mentioned origin O_m . Especially, the origin O_m of the camera coordinate system with known world coordinates (X_0, Y_0, Z_0) just appears to be the image center O_c with known image coordinates (u_0, v_0) . Let the image point p_k at coordinates (u_k, v_k) with respect to the origin O_c of the image coordinate system (ICS) and the world-space point P_k at coordinates (X_k, Y_k, Z_k) with respect to the origin O_m of the corresponding world coordinate system (WCS) form a landmark point pair. Also, assume that n sets of landmark point pairs (p_k, P_k) are selected, where $k = 0, 1, \dots, n - 1$.

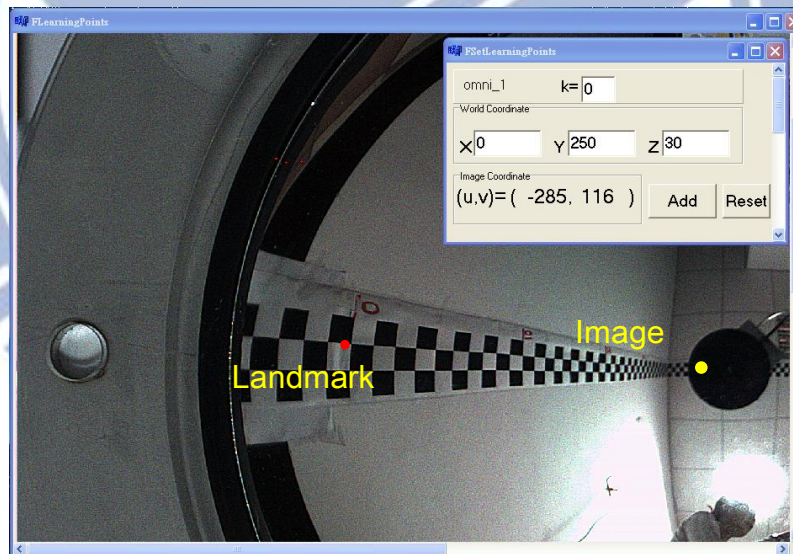


Figure 6.3 The interface for acquiring the data of the world space points.

2. Estimation of coefficients of radial stretching function

Due to the nonlinear property of the hyperbolic mirror surface shape, the radial-directional mapping should be specified by a nonlinear function f_r . More specifically, Figure 6.4 shows that each of the elevation angles corresponds to a radial

distance, or by notations, that the elevation angle ρ of each world point P corresponds to the radius distance r of its corresponding image point p . Therefore, the radial distance r from each image pixel p at coordinates (u, v) in the omni-image to the image center O_c at coordinates (u_0, v_0) may be computed by $r = f_r(\rho)$. In this study, we call the function f_r the *radial stretching function* of the omni-camera and approximate it by the following 5th-degree polynomial function:

$$r = f_r(\rho) = a_0 + a_1 \times \rho^1 + a_2 \times \rho^2 + a_3 \times \rho^3 + a_4 \times \rho^4 + a_5 \times \rho^5 \quad (6.1)$$

where a_0 through a_5 are six coefficients to be estimated using the n landmark point pairs, as described in the following algorithm [4].

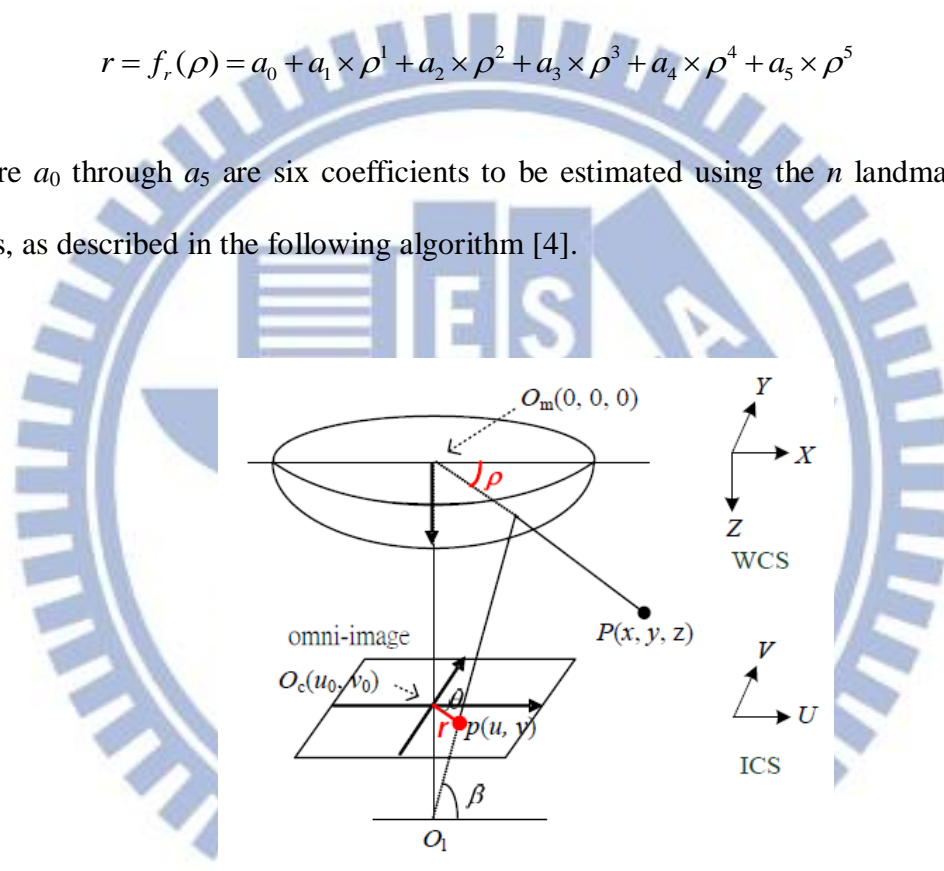


Figure 6.4 Nonlinear property of an omni-camera with mirror surface shape.

Algorithm 6.1 Estimation of the coefficients of the radial stretching function.

Step 1. (*Elevation angle and radial distance calculation*) Use the coordinate data of landmark point pair (P_k, p_k) , including (X_k, Y_k, Z_k) in the WCS and (u_k, v_k) in the ICS, to calculate the elevation angle ρ_k of P_k in the world space and the radial distance r_k of ρ_k in the ICS by the following equations:

$$\rho_k = \tan^{-1}\left(\frac{Z_k}{D_k}\right); \quad r_k = \sqrt{u_k^2 + v_k^2} \quad (6.2)$$

where D_k , is the distance from the landmark point P_k to the mirror center O_m in the X - Y plane of the WCS, computed by $D_k = \sqrt{X_k^2 + Y_k^2}$.

Step 2. (*Calculation of the coefficients of the radial stretching function*) Substitute all the data $\rho_0, \rho_1, \dots, \rho_{n-1}$ and r_0, r_1, \dots, r_{n-1} into Equation (3.1) to get n homogeneous equations as follows:

$$\begin{aligned} r_0 &= f_r(\rho_0) = a_0 + a_1 \times \rho_0^1 + a_2 \times \rho_0^2 + a_3 \times \rho_0^3 + a_4 \times \rho_0^4 + a_5 \times \rho_0^5; \\ r_1 &= f_r(\rho_1) = a_0 + a_1 \times \rho_1^1 + a_2 \times \rho_1^2 + a_3 \times \rho_1^3 + a_4 \times \rho_1^4 + a_5 \times \rho_1^5; \\ &\vdots \\ r_{n-1} &= f_r(\rho_{n-1}) = a_0 + a_1 \times \rho_{n-1}^1 + a_2 \times \rho_{n-1}^2 + a_3 \times \rho_{n-1}^3 + a_4 \times \rho_{n-1}^4 + a_5 \times \rho_{n-1}^5; \end{aligned} \quad (6.3)$$

and solve the equations to get the desired coefficients ($a_0, a_1, a_2, a_3, a_4, a_5$) of the radial stretching function f_r by a numerical analysis method.

3. Pano-mapping table construction

Each entry E_{ij} with indices (i, j) in the pano-mapping table specifies an azimuth-elevation angle pair (θ_i, ρ_j) as shown in Figure 6.5, which represents an infinite set S_{ij} of points in the world space passing through by the light ray with azimuth angle θ_i and elevation angle ρ_j . These world space points in S_{ij} are all projected onto an identical pixel p_{ij} in an omni-image taken by the camera, forming a *pano-mapping* f_{pm} from S_{ij} to p_{ij} .

Table 6.1 is shown an example of the pano-mapping table by filling entry E_{ij} with the coordinates (u_{ij}, v_{ij}) of pixel p_{ij} in the omni-image. A pano-mapping table T_{pm} of $M \times N$ entries is created by dividing the range 2π ($= 360^\circ$) of the azimuth angles into M units as well as dividing the range of the elevation angles from ρ_s to ρ_e into N units.

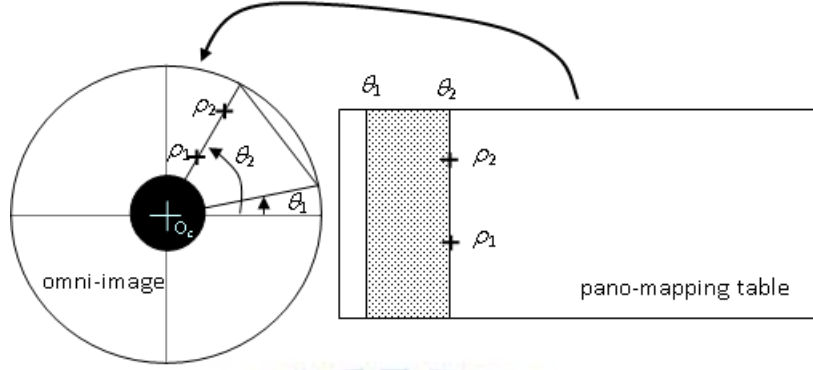


Figure 6.5 Mapping between pano-mapping table and omni-image.

According to the rotation-invariant property of the omni-camera, the azimuth angle θ of a world point P which the light ray passes through is essentially identical to the angle ϕ of the corresponding pixel p with respect to the u -axis in the input image I . That is, the azimuth-directional mapping denoted is an identity function f_a such that $f_a(\theta) = \phi = \theta$. Accordingly, the entries of table T_{pm} may be filled by the following algorithm.

Algorithm 6.2 Creation of the pano-mapping table.

Step 1. Divide the range 2π of the azimuth angles into M intervals, and compute the i th azimuth angle θ_i by

$$\theta_i = i \times (2\pi / M), \text{ for } i = 0, 1, \dots, M - 1.$$

Step 2. Divide the range $[\rho_s, \rho_e]$ of the elevation angles into N intervals, and estimate the j th elevation angle ρ_j by

$$\rho_j = j \times [(\rho_e - \rho_s) / N] + \rho_s, \text{ for } j = 0, 1, \dots, N - 1.$$

Step 3. Regard the pairs $(r_j, \phi_i) = (f_r(\rho_j), \theta_i)$ of all the image pixels to form a polar coordinate system with the image coordinates (u, v) specified by

$$u_{ij} = r_j \times \cos \phi_i = f_r(\rho_j) \times \cos \theta_i;$$

$$v_{ij} = r_j \times \sin \phi_i = f_r(\rho_j) \times \sin \theta_i,$$

and fill the entry E_{ij} with the corresponding image coordinates (u_{ij}, v_{ij}) where

$$r_j = f_r(\rho_j) = a_0 + a_1 \times \rho_j^1 + a_2 \times \rho_j^2 + a_3 \times \rho_j^3 + a_4 \times \rho_j^4 + a_5 \times \rho_j^5,$$

with the coefficients $(a_0, a_1, a_2, a_3, a_4, a_5)$ being computed by Algorithm 6.1.

Finally, we use the above-mentioned method to create a pano-mapping tables with size $M \times N$. The elevation angle ranges of the field of view of the omni-images are from ρ_{us} to ρ_{ue} , respectively. The elevation angle ρ of the light ray horizontally going through the focal point of the mirror is 0° , so ρ_{us} as well as ρ_{ls} is negative and ρ_{ue} as well as ρ_{le} is positive.

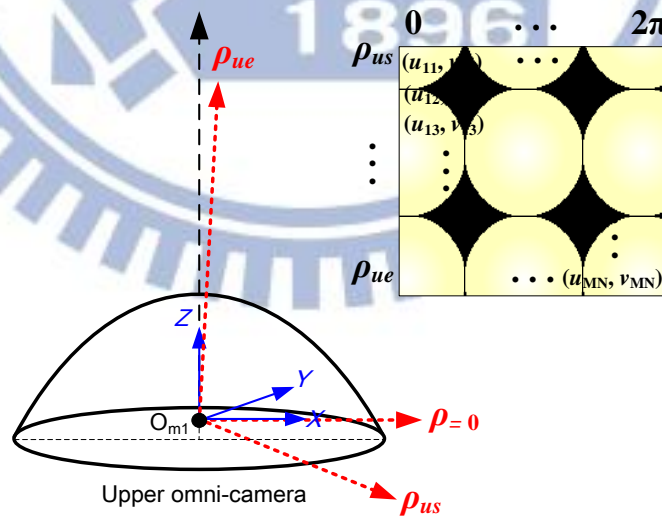


Figure 6.6 Creation of pano-mapping table.

Steps:

Step 1. As illustrated in Figure 6.1, calculate the angle ϕ according to trigonometry to be as follows:

$$W^2 = D^2 + D^2 - 2 \times D \times D \times \cos \phi, \quad (6.4)$$

or equivalently, as follows:

$$\phi = \cos^{-1} \left[1 - \frac{W^2}{2 \times D^2} \right]. \quad (6.5)$$

Step 2. Calculate the angle β shown in Figure 6.7 according to trigonometry to be as follows:

$$\beta = \frac{\pi - \phi}{2}. \quad (6.6)$$

Step 3. Compute the index i of entry E_{ij} of table T_{pm} corresponding to pixel q_{kl} in image Q at coordinates (k, l) in the following way.

3.1 Let P_{ij} denote the intersection point of the light ray R_q projected onto q_{kl} and the planar projection region A_p , and compute the distance d between point P_{ij} and the border point P_r shown in Figure 6.7 by linear proportionality as:

$$d = k \times \frac{W}{M_Q}, \quad (6.7)$$

where the projection region A_p has a width of W , the image Q has a width of M_Q pixels, and pixel q_{kl} has an index of k in the horizontal direction.

3.2 Compute the distance L between point P_{ij} and the mirror center O_m according to trigonometry as follows:

$$L = \sqrt{D^2 + d^2 - 2 \times d \times D \times \cos \beta}. \quad (6.8)$$

3.3 Compute the distance h from point P_{ij} to the line segment $O_m P_r$ connecting O_m and P_r as:

$$h = d \times \sin \beta. \quad (6.9)$$

3.4 Compute the azimuth θ_q of point P_{ij} with respect to $O_m P_r$ satisfying:

$$\sin \theta_q = \frac{h}{L} = \frac{d \times \sin \beta}{\sqrt{D^2 + d^2 - 2 \times d \times D \times \cos \beta}}, \quad (6.10)$$

which leads to

$$\theta_q = \sin^{-1} \frac{h}{L} = \sin^{-1} \left[\frac{d \times \sin \beta}{\sqrt{D^2 + d^2 - 2 \times d \times D \times \cos \beta}} \right]. \quad (6.11)$$

3.5 Compute the index i of entry E_{ij} by linear proportionality as:

$$i = \frac{\theta_q}{2\pi} \times M. \quad (6.12)$$

Step 4. Compute the index j of entry E_{ij} of table T_{pm} corresponding to pixel q_{kl} in image Q at coordinates (k, l) in the following way.

4.1 As shown in Figure 6.8 which is the involved imaging configuration from a lateral view, let the height of the projection region A_p be H , divide the image Q into N_Q intervals, and compute the height of P_{ij} by linear proportionality again to be:

$$H_q = l \times \frac{H}{N_Q}. \quad (6.13)$$

4.2 Compute the elevation angle ρ_q according to trigonometry as:

$$\rho_q = \tan^{-1}\left(\frac{H_q}{L}\right). \quad (6.14)$$

4.3 Compute the index j of E_{ij} by proportionality again to be:

$$j = \frac{(\rho_q - \rho_s) \times N}{(\rho_e - \rho_s)}. \quad (6.15)$$

Step 5. Obtain the coordinates (u_{ij}, v_{ij}) in G with the indices (i, j) of E_{ij} .

Step 6. Assign the color value of the image pixel p_{ij} of G at coordinates (u_{ij}, v_{ij}) to pixel q_{kl} of Q at coordinates (k, l) .

Step 7. After all pixels of Q are processed, take the final content of Q as the desired perspective-view image.

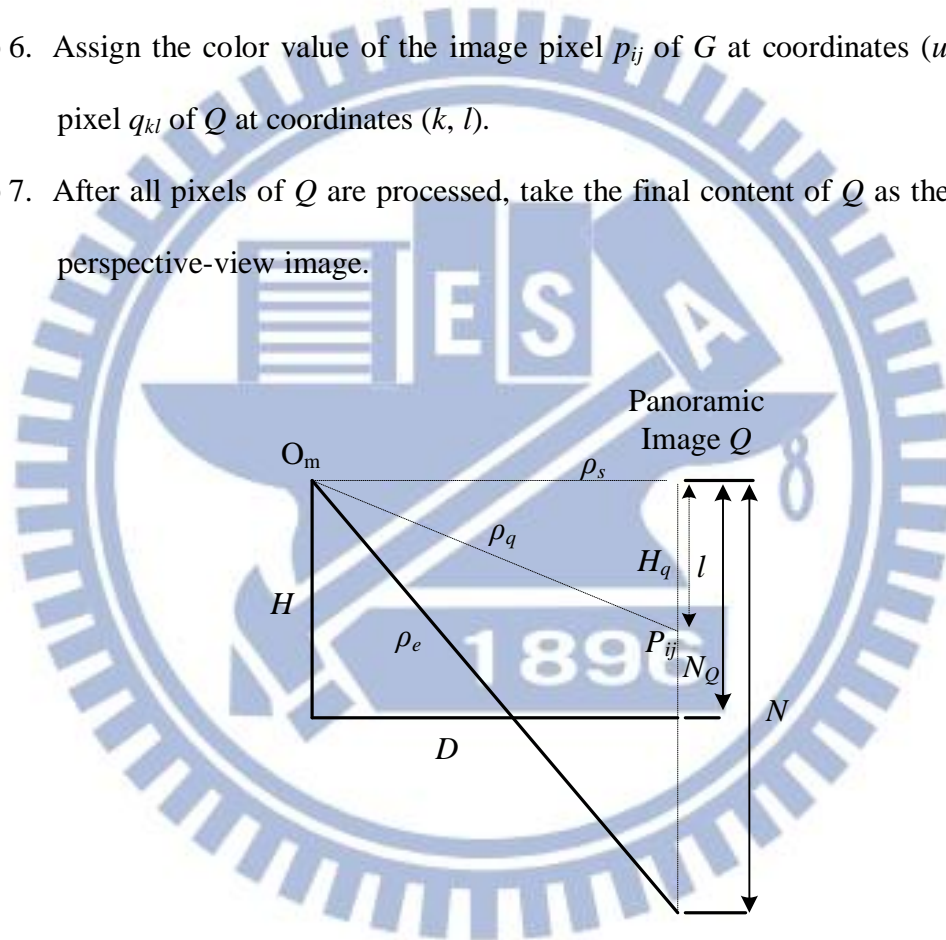


Figure 6.8 A lateral-view configuration for generating a perspective-view image.

6.2.3 Review of Generation of Perspective-mapping

Table

According to the above-mentioned method for unwarping omni-images into

perspective-view images, we can obtain the perspective-view images of the upper and lower omni-images. However, a great amount of computation is repeated, as can be seen from Equation 6.4 through 6.15, resulting in possible delays of image display in the scene browsing system. Therefore, in order to avoid such massive computation, it was found possible to establish a table in advance for an omni-camera to generate perspective-view images in a faster way. The table is called a *perspective-mapping table*. Such a table may be created from any view direction.

Specifically, given an omni-image G and a pano-mapping table T_{pm} with $M \times N$ entries, we generate the perspective-mapping table according to the algorithm described in the following.

Algorithm 6.4: Generation of a perspective-mapping table.

Input: an omni-image G and a pano-mapping table T_{pm} with $M \times N$ entries.

Output: a perspective-mapping table T_{pe} .

Steps.

- Step 1. Decide the size of a *maximum* perspective-view image Q to be generated, say, $M_p \times N_p$.
- Step 2. Establish an *empty* perspective-mapping table T_{pe} with the same size.
- Step 3. Let θ specify the “direction” for table T_{pe} , which is also the view direction from which the perspective-view image Q is to be generated.
- Step 4. Let ρ_e and ρ_s be the maximum and the minimum elevation angles used in constructing the pano-mapping table T_{pm} .
- Step 5. Define the size of a planar rectangular region A_p to be $W \times H$, which describe the maximum size of a perspective-view image to be generated.
- Step 6. Select a value D as the distance of A_p to the mirror center O_m , which is also the distance of the perspective-view image to O_m .

Step 7. Compute H of A_P according to the trigonometry as illustrated in Figure 6.9 by the following equation:

$$H = D \times \tan \rho_e + |D \times \tan \rho_s|. \quad (6.16)$$

Step 8. Take θ be zero, as shown in Figure 6.9(a).

Step 9. Apply Algorithm 6.3 to the omni-image G using pano-mapping table T_{pm} with $M \times N$ entries and H as input, to generate a perspective-view image I .

Step 10. Record the corresponding indices (i, j) in T_{pm} of all pixels in I into the entries S_{mn} with indices (m, n) of T_{pe} .

Step 11. After all entries of T_{pe} are filled, take the final content of T_{pe} as the desired perspective mapping table.

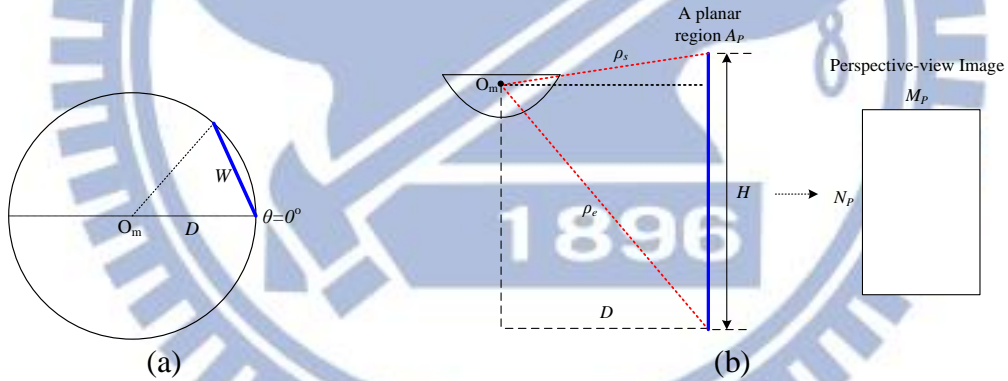


Figure 6.9 Illustration of construction of a perspective-mapping table. (a) A Top-view configuration for generating a perspective-mapping table. (b) A lateral-view configuration for generating a perspective-view image.

6.2.4 Generation of Passenger-view Image

After establishing the perspective-mapping table T_{pe} , we can use it to generate perspective-view images. In this study, we use the passenger-view image as the base of the AR image. A user in the vehicle can see the front passenger's views through the

windows on the mobile-device screen. In order to generate the passenger-view image, we have to measure some geometric information about the interior of the vehicle. Firstly, we assume that the viewpoint originates from the front passenger seat as shown in Figure 6.10, and measure the angles involved in the views that can be seen through the windows from the viewpoint as shown in Figure 6.10.

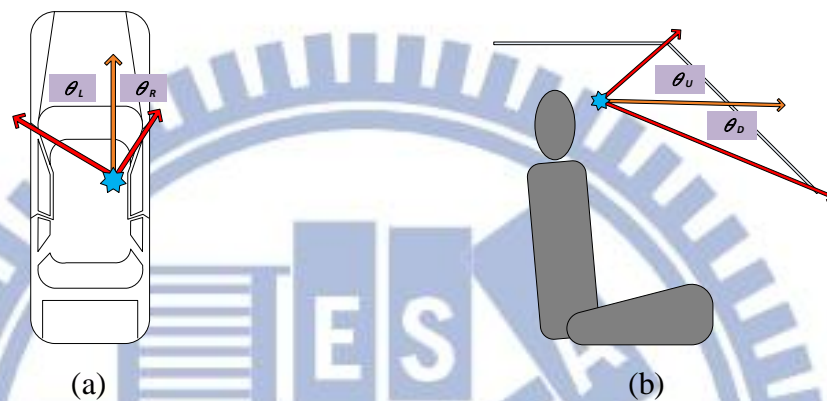


Figure 6.10 An illustration of viewpoint in the vehicle. (a) Top-view of the vehicle where the blue star is the viewpoint and the red line is the region of the view. (b) Side-view of the vehicle where again the blue star is the viewpoint and the red line is the region of view.

In more detail, the region covered by the view, called *view region*, is not a rectangle; instead, it is a quadrilateral, as shown in Figure 6.11(e). So we compute the positions of its four vertices. And then, we use the positions of the vertices to cut a part of the image which just fits the passenger's view. Note that the viewpoint we set is not the same as the camera. So we shift the image coordinate system to move the viewpoint of the camera to the previously-mentioned front-passenger's viewpoint as illustrated in Figure 6.12. The following algorithm is proposed for generating the passenger-view image on the mobile-device screen.

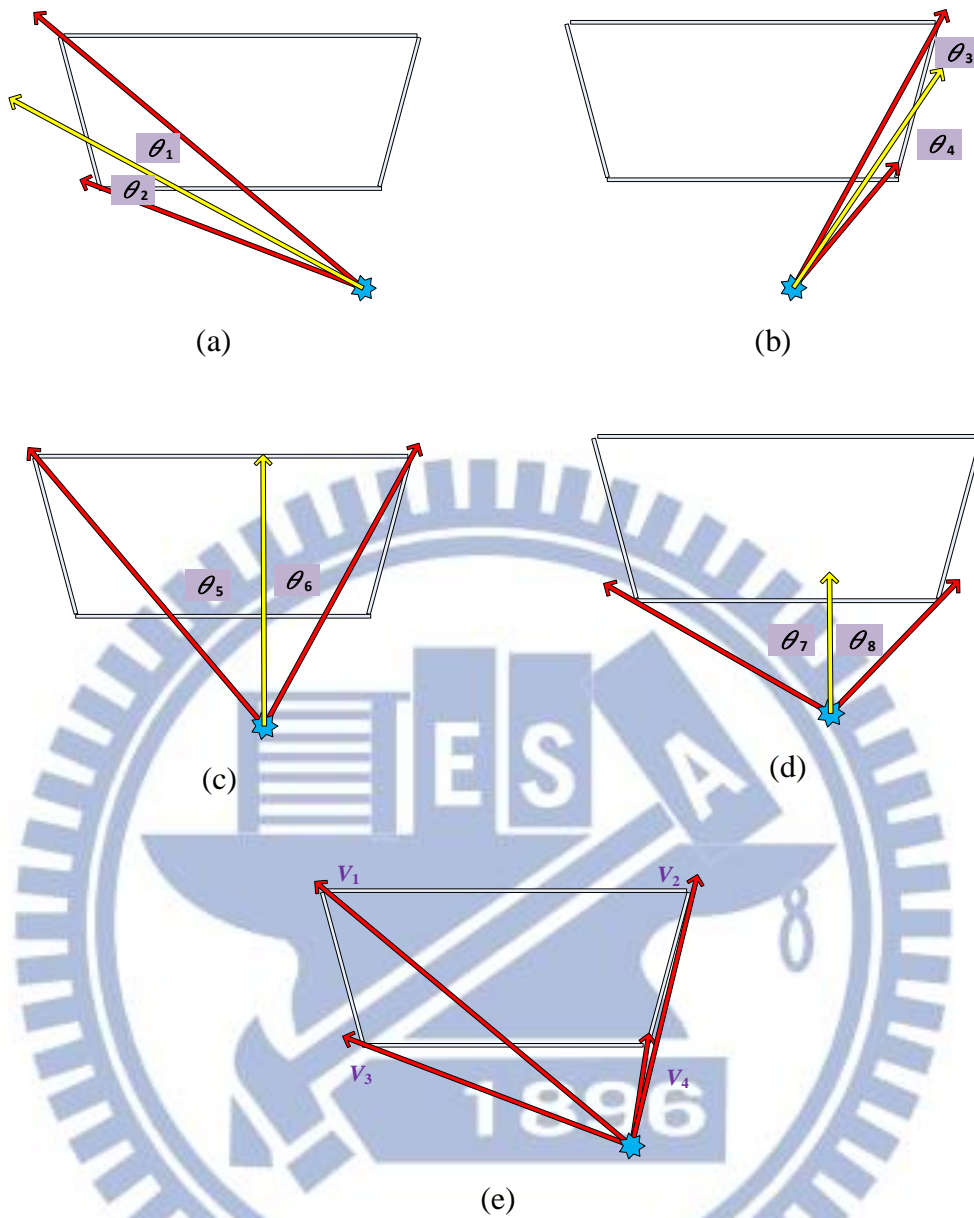


Figure 6.11 An illustration of viewpoints through the windshield. (a) The left side angles, where the yellow line is a horizontal line with an angle of zero, and the red line is the boundary of the viewpoint. (b) The right side angles. (c) The upside angles, where the yellow line is a vertical line with an angle of zero, and the red line is the boundary of the viewpoint. (d) The downside angles. (e) all view of the viewpoint.

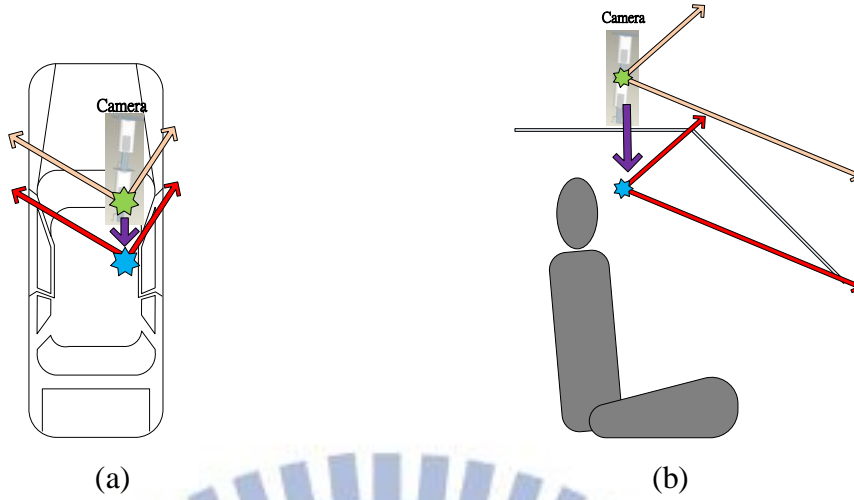


Figure 6.12 An illustration of shifting the viewpoint. (A) Top-view of shifting the viewpoint where the blue star is the viewpoint we set and the green star is the viewpoint of camera. (B) Side-view of shifting the viewpoint where the blue star is the viewpoint we set and the green star is the viewpoint of camera..

Algorithm 6.5 Generation of the passenger-view image.

Input: an omni-image G , a perspective-mapping table T_{pe} with $M_P \times N_P$ entries, the center point $C(x_c, y_c)$ of the image, and the pre-measured angles θ_i of the view region as shown in Figure 6.11.

Output: a passenger-view image I_p to be displayed on the user's mobile-device screen.

Steps:

Step 1. Compute the position of the vertex $V_i(x_i, y_i)$ of the passenger's view image according to the perspective-mapping table, Table T_{pe} , by the following equation according to the principle of proportionality:

$$\begin{aligned} x_i &= x_c + \frac{\theta_x}{360} \times M_P; \\ y_i &= y_c + \frac{\theta_y}{110} \times N_P, \end{aligned} \tag{6.17}$$

where θ_x is the angle in the horizontal direction and θ_y is the angle in the

vertical direction as shown in Figures 6.13(a) and (b) (note that the camera has 360 degrees of view in horizontal direction and 100 degrees of view in the vertical direction).

- Step 2. Repeat Step 1 until the positions of the four vertices are all computed.
- Step 3. Use the perspective-mapping table T_{pe} to draw the desired image I_p as shown in Figure 6.13 by the following steps.
- 3.1 Set the shift index $S(x_s, y_s)$ to be the position of the upper left vertex of V_i .
 - 3.2 Set print point $P(x_p, y_p)$ to be $(0, 0)$.
 - 3.3 According to the coordinates $(x_s + x_p, y_s + y_p)$ in the table T_{pe} , find the corresponding image coordinates (u, v) .
 - 3.4 Assign the color value of each image pixel of G at coordinates (u, v) to a pixel at coordinates (x_p, y_p) in the desired passenger's view image I_p .
 - 3.5 Move P to the next point.
 - 3.6 Repeat Steps 3.3 to 3.5 until all the pixels in the region of passenger-view image are drawn in I_p .

6.3 Augmenting Names of Buildings on Passenger-view Images

After generating the passenger-view image, the system can use this image as the base to augment the building name on it. In order to implement this, we need to compute the information about the position of the building by the use of the learned data and the location of the vehicle. The proposed method to augment the building name is described in the next section.

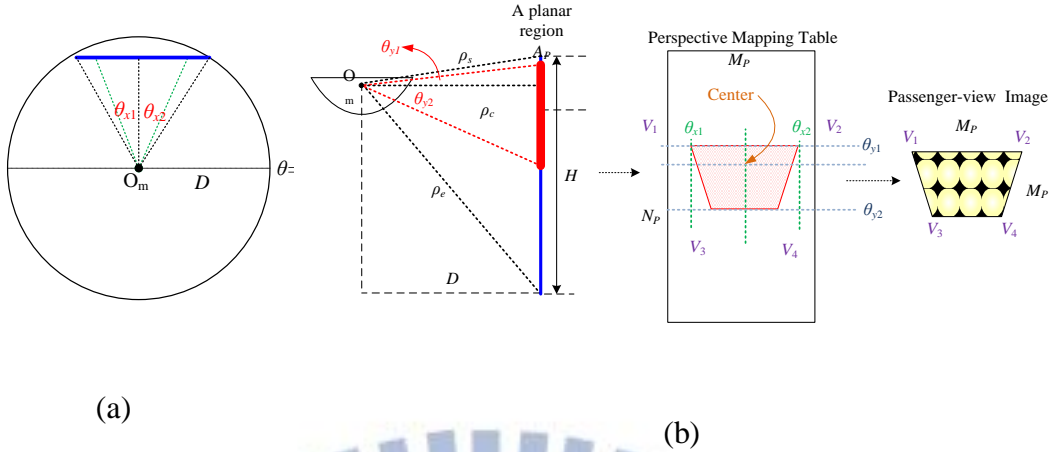


Figure 6.13 Illustration of construction of passenger-view images. (a) A Top-view configuration for generating a passenger-view image. (b) A lateral-view configuration for generating a passenger-view image.

6.3.1 Calculating Positions of Buildings in Passenger-view Images

We calculate the position of the building by using the result of vehicle localization. Then, we can use the position to calculate the angle of the direction of the building as seen from the vehicle, as illustrated in Figure 6.14. Note that the building appears to be an area instead of a point. Thus, we calculate three angles of from the vehicle to two corners and the middle of the building by Equation (3.2) as shown in Figure 6.14. Equation (3.2) is repeated in the following for convenience of reference:

$$\theta_p = \tan^{-1}\left(\frac{x_1 - x_2}{y_1 - y_2}\right). \quad (3.2)$$

Moreover, we use the angle to calculate the building position in the image. In more detail, we know the position and the corresponding angle in the passenger-view image after generating the image. Then, we just use the corresponding relation to

compute the position of the building as shown in Figure 6.15. Finally, the system uses the learned data to augment the correct building name on the passenger-view image.

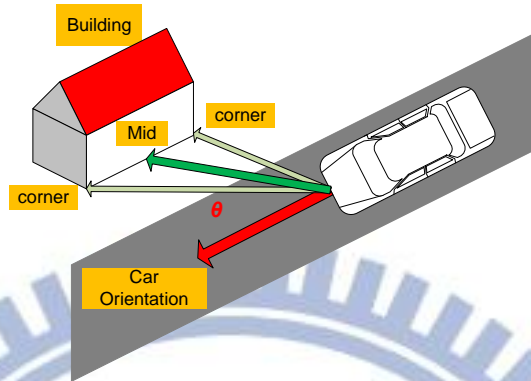


Figure 6.14 An illustration for calculating the angle of the direction to the building.

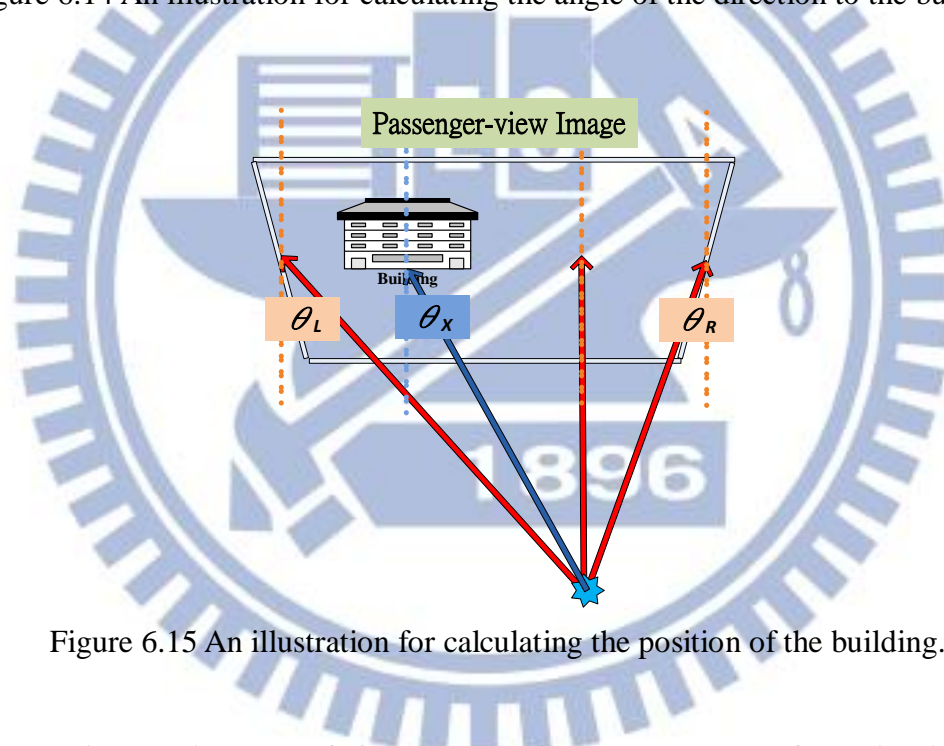


Figure 6.15 An illustration for calculating the position of the building.

6.3.2 Algorithm of Augmenting Names of Buildings on Images

In this section, we introduce the proposed algorithm of augmenting the names of buildings on images. After calculating the position of the building in the passenger-view image, we can use the learned data of the building to augment the building information on the image. However, we regard the building as a line in this

study. In more detail, we learn the edge line of the building by connecting two corners of it as shown in Figure 6.16. In some cases, the entire building is not seen in the image; instead, only part of the building can be seen in the image as shown in Figure 6.17(b). In one case, the middle part of the building is in the image, and we can augment the building name on it. In another case, only part of the building can be seen in the image; then, we augment the building name on the boundary of the image. The detail is described in the following as an algorithm.

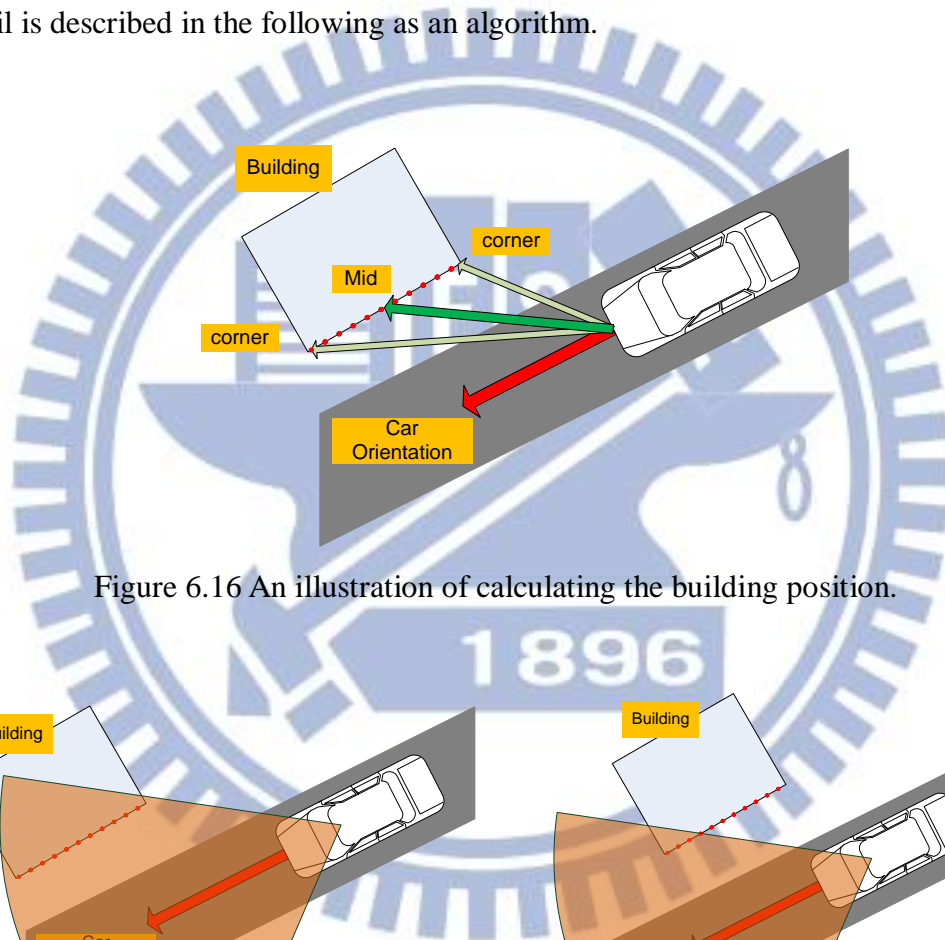


Figure 6.16 An illustration of calculating the building position.

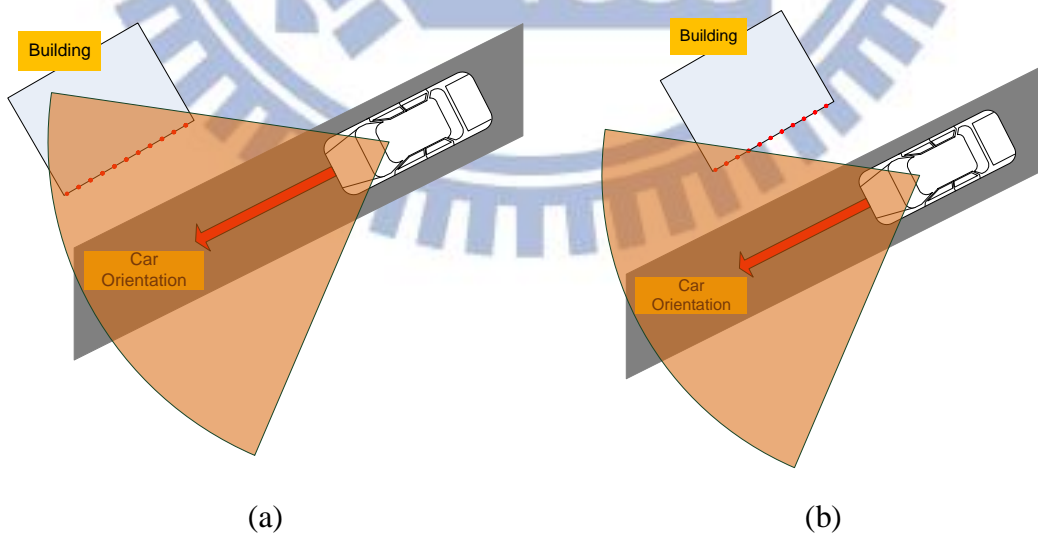


Figure 6.17 An illustration of the view of the image. (a) The entire building can be seen in the image. (b) Only part of the building can be seen in the image.

Algorithm 6.6 Augmenting Names of Buildings on the Images.

Input: the position P_v of the vehicle, a passenger-view image I_p , the width W and height H of I_p , the center point $C(x_c, y_c)$ of the image, and the angles θ_L and θ_R of the left boundary and the right boundary of the passenger-view image I_p , respectively.

Output: the passenger-view image I_{AR} with augmented information.

Steps:

- Step 1. Set the angle θ_f of the direction of the building in I_p to be zero.
- Step 2. Use the position P_v of the vehicle to compute the angle θ_{BM} of the direction of the middle of the building in I_p by Equation 3.2 and Equation 3.3.
- Step 3. Use P_v to compute the angle θ_{BL} of the direction of the left side of the building in I_p by Equation 3.2 and Equation 3.3.
- Step 4. Use P_v to compute the angle θ_{BR} of the direction of the right side of the building in I_p by Equations 3.2 and 3.3.
- Step 5. If the following equation is satisfied, set θ_f to be θ_{BM} and go to Step 8; else, go to Step 6:

$$\theta_L \leq \theta_{BM} \leq \theta_R. \quad (6.18)$$

- Step 6. If the following equations are satisfied, set θ_f to be θ_L and go to Step 8; else, go to Step 7:

$$\begin{aligned} \theta_{BM} &< \theta_L; \\ \theta_L &\leq \theta_{BR} \leq \theta_R. \end{aligned} \quad (6.19)$$

- Step 7. If following equation is true, set θ_f to be θ_R and go to step 8.

$$\begin{aligned} \theta_R &< \theta_{BM}; \\ \theta_L &\leq \theta_{BL} \leq \theta_R. \end{aligned} \quad (6.20)$$

- Step 8. Compute the position $P(x, y)$ of the building by the following equation:

$$\begin{aligned} x &= \frac{\theta_f}{\theta_R - \theta_L} \times W + x_c; \\ y &= y_c. \end{aligned} \quad (6.21)$$

Step 9. Use the learned data to augment the name of building at position $P(x, y)$ on image I_p , and save it into I_{AR} .

6.4 Tour Guidance in Park Areas

6.4.1 Ideas of Tour Guidance in Park Areas

By augmenting relevant information on the passenger-view image for inspection on the user's mobile-device screen, the system can give the user a good guidance in a park area. Moreover, the system uses many methods described in Chapters 3 through 6. Firstly, the system learns the environment for constructing the environment map in the park area. Secondly, the system detects the along-path line features for locating the vehicle. Also, the system uses the features to locate the vehicle. Furthermore, the system uses the location of the vehicle to compute the position of the building. Finally, the system shows the augmented information on the passenger-view image generated by the system. When the vehicle is driven in the park area, the user can enjoy the tour guidance by all these functions!

6.4.2 Algorithm for Tour Guidance in Park Areas

In this section, we introduce the tour guidance algorithm which uses all the methods described in the previous chapters. The algorithm is described in the following.

Algorithm 6.7 Tour guidance in park area.

Input: an environment map Map , a color omni-image I_c , a set of positions of center points P_i , a threshold g of gap size, and threshold w of the line width, a reference omni-image I_r , a search window W , two threshold values T_u and T_d , a threshold d for judging the equality of two angles, perspective-mapping table T_{pe} with $M_P \times N_P$ entries, a center point $C(x_c, y_c)$ of the image, and angles θ_L and θ_R of the left boundary and the right boundary of the passenger-view image.

Output: a passenger-view image I_{AR} with augmented information.

Steps:

- Step 1. Transform the color omni-image I_c into a gray-level omni-image I_g .
- Step 2. Transform the gray-level omni-image I_g into a binary omni-image I_b .
- Step 3. With the binary image I_b , a set of positions of center points P_i , a threshold g of the gap size, and the threshold w of line width as inputs, perform Algorithm 4.3 to conduct detection of lines in a set S_i .
- Step 4. With the environment map Map , a set S_i , a search window W , two threshold values T_u and T_d as inputs, perform Algorithm 5.6 to compute the position P_v of the vehicle.
- Step 5. With the omni-image I_c and the pano-mapping table T_{pm} with $M \times N$ entries as inputs, perform Algorithm 6.5 to generate the passenger-view image I_p .
- Step 6. With the position P_v of the vehicle, a passenger-view image I_p , and the angles θ_L and θ_R as inputs, perform Algorithm 6.6 to generate the passenger-view image I_{AR} with the building name augmented on it.
- Step 7. Display I_{AR} on the mobile-device screen.

6.5 Experimental Results

We have described the proposed methods for augmenting the building name on the passenger-view image. As described in Section 6.2, we generate the passenger-view image by transforming the omni-image. An example of the generated passenger-view image is shown in Figure 6.18. As described in Section 6.3, we augment the building name on the passenger-view image. An example of augmenting the building name on the passenger-view image is shown in Figure 6.19.

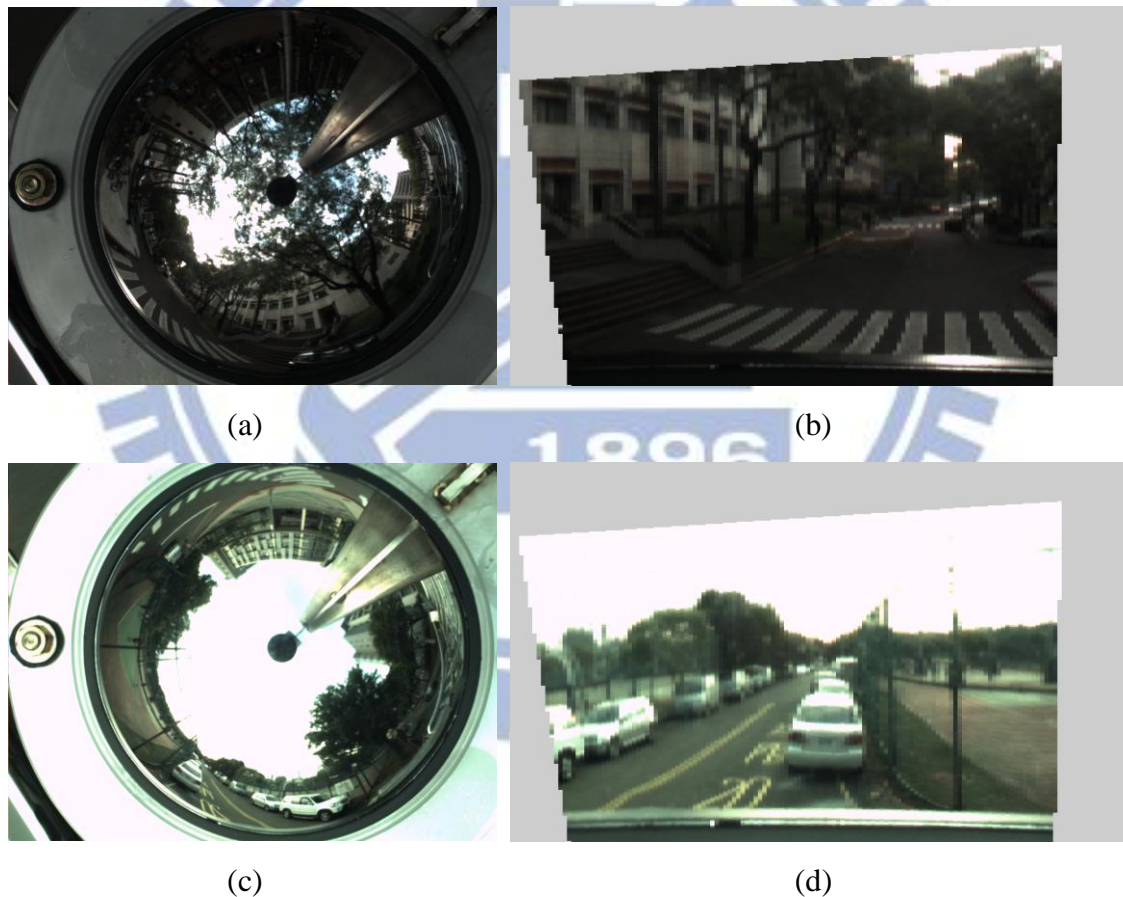
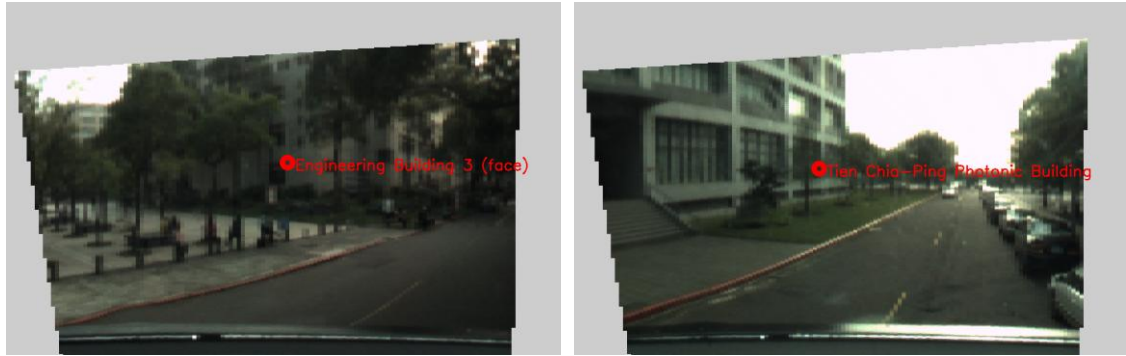


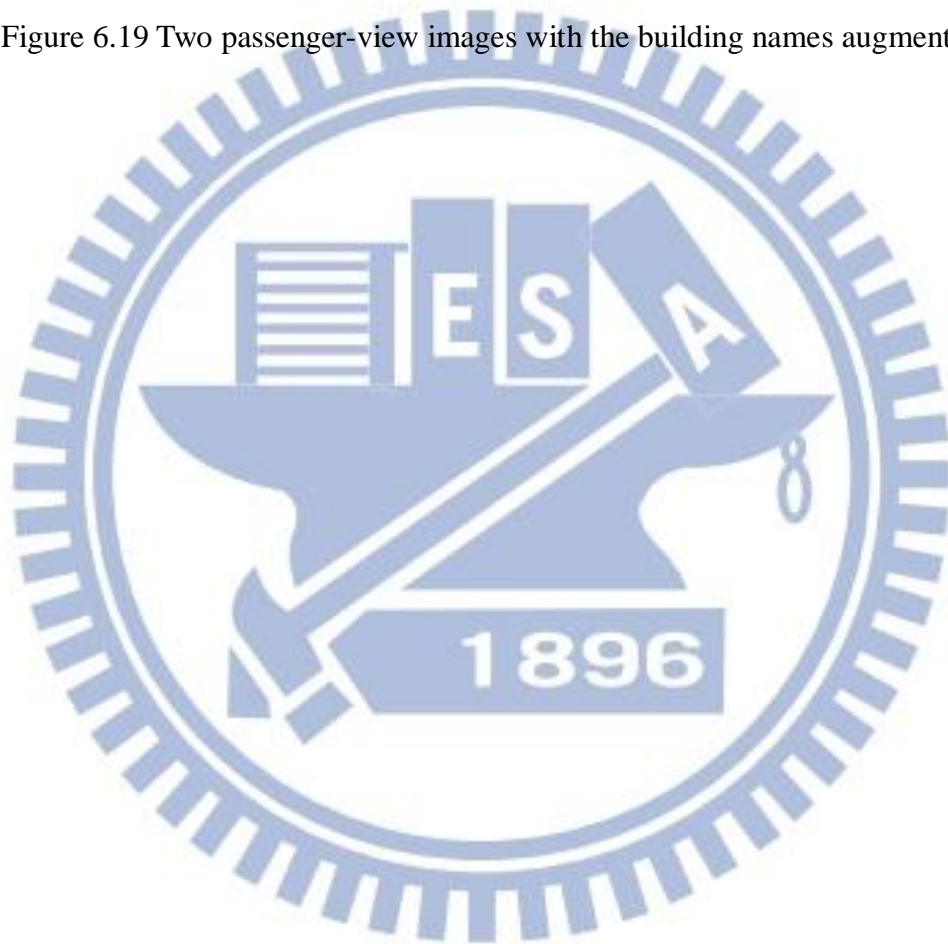
Figure 6.18 (a)The omni-image acquired from the omni-camera. (b)The passenger-view image transformed from (a). (c)The omni-image acquired from the omni-camera. (d)The passenger-view image transformed from (c)



(a)

(b)

Figure 6.19 Two passenger-view images with the building names augmented.



Chapter 7

Experimental Results and Discussions

7.1 Experimental Results

In this chapter, we will show some experimental results of applying the proposed augmented reality guidance system for park touring on the vehicle. We will show the results of learning the environment map for our experimental environment, which is part of the National Chiao Tung University campus, and the results of the guidance process based on the AR technique developed for this study. An illustration of the guidance area consisting of a path, four buildings, ten light poles, and seventeen edge lines on building walls along the sidewalk is shown in Figure 7.1.

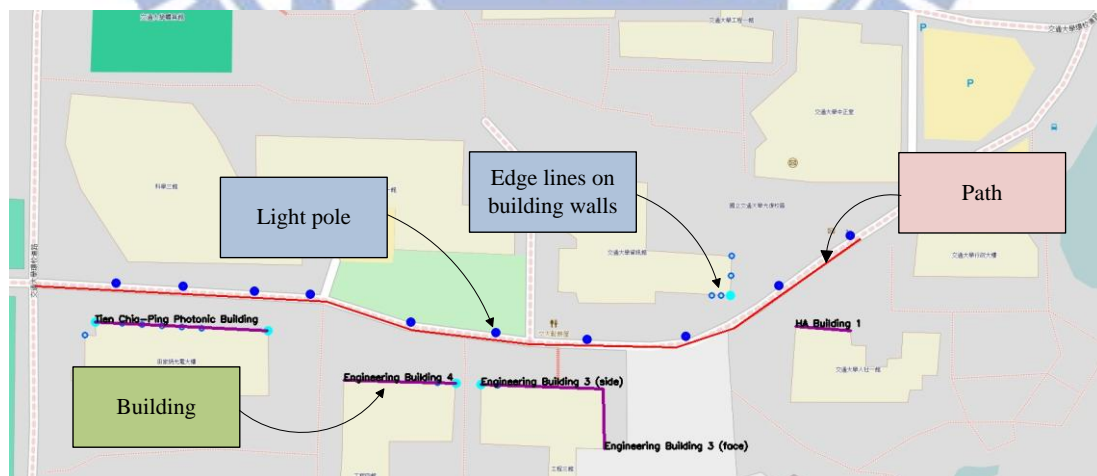


Figure 7.1 The environment map we use in the proposed system.

In the learning stage, we drive the vehicle on the path as shown in Figure 7.2(a). When the system detects vertical lines, we save the information into the database as shown in Figure 7.2(b) and (c).

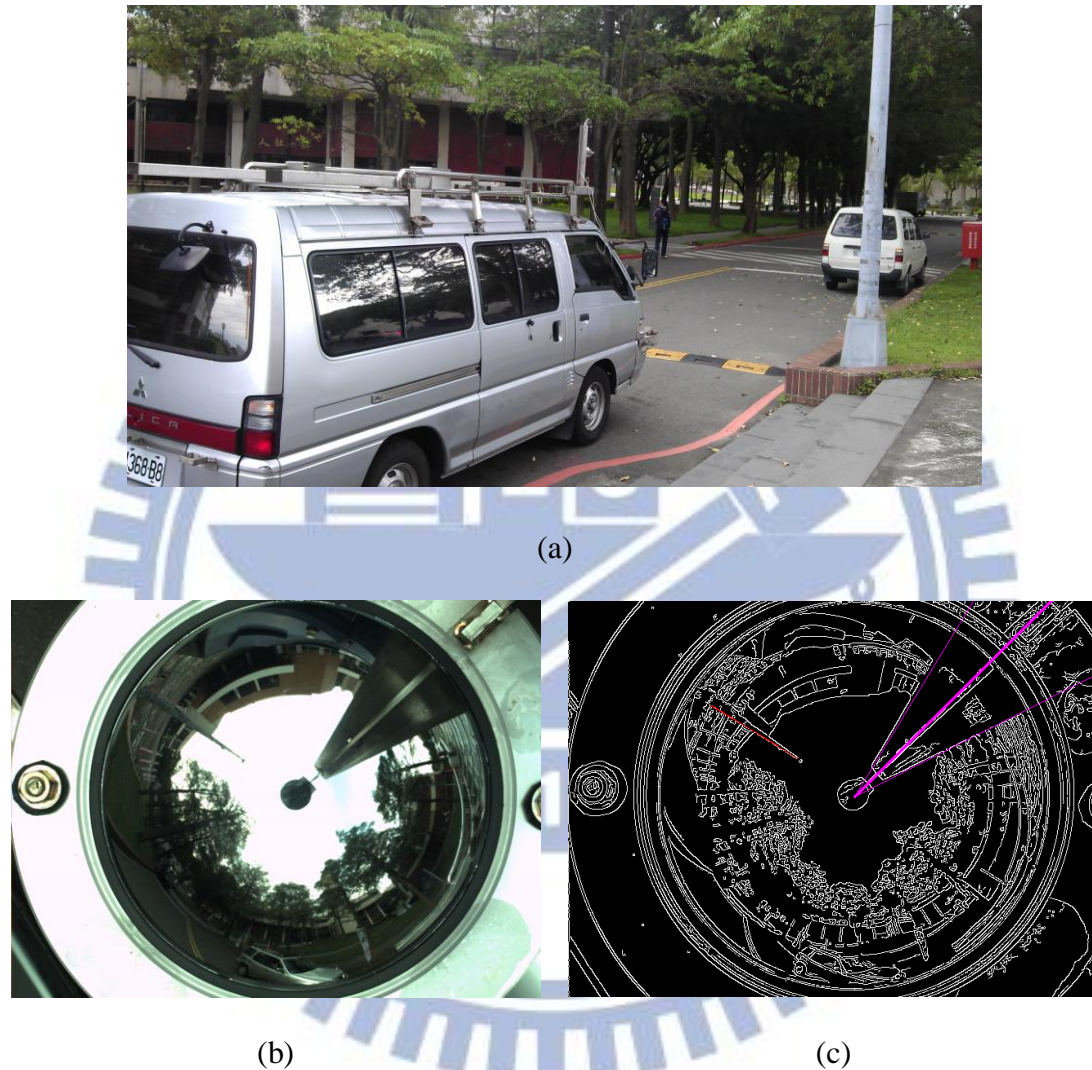


Figure 7.2 An experimental result of the learning stage. (a) An image of the vehicle driven on the path and detecting the line feature. (b) An omni-image acquired from the omni-camera. (c) A line feature detected by the system.

After learning the environment, the system can use the learned data to offer AR-based guidance to the user. Firstly, the system detects the vertical line features for vehicle localization as shown in Figure 7.3.

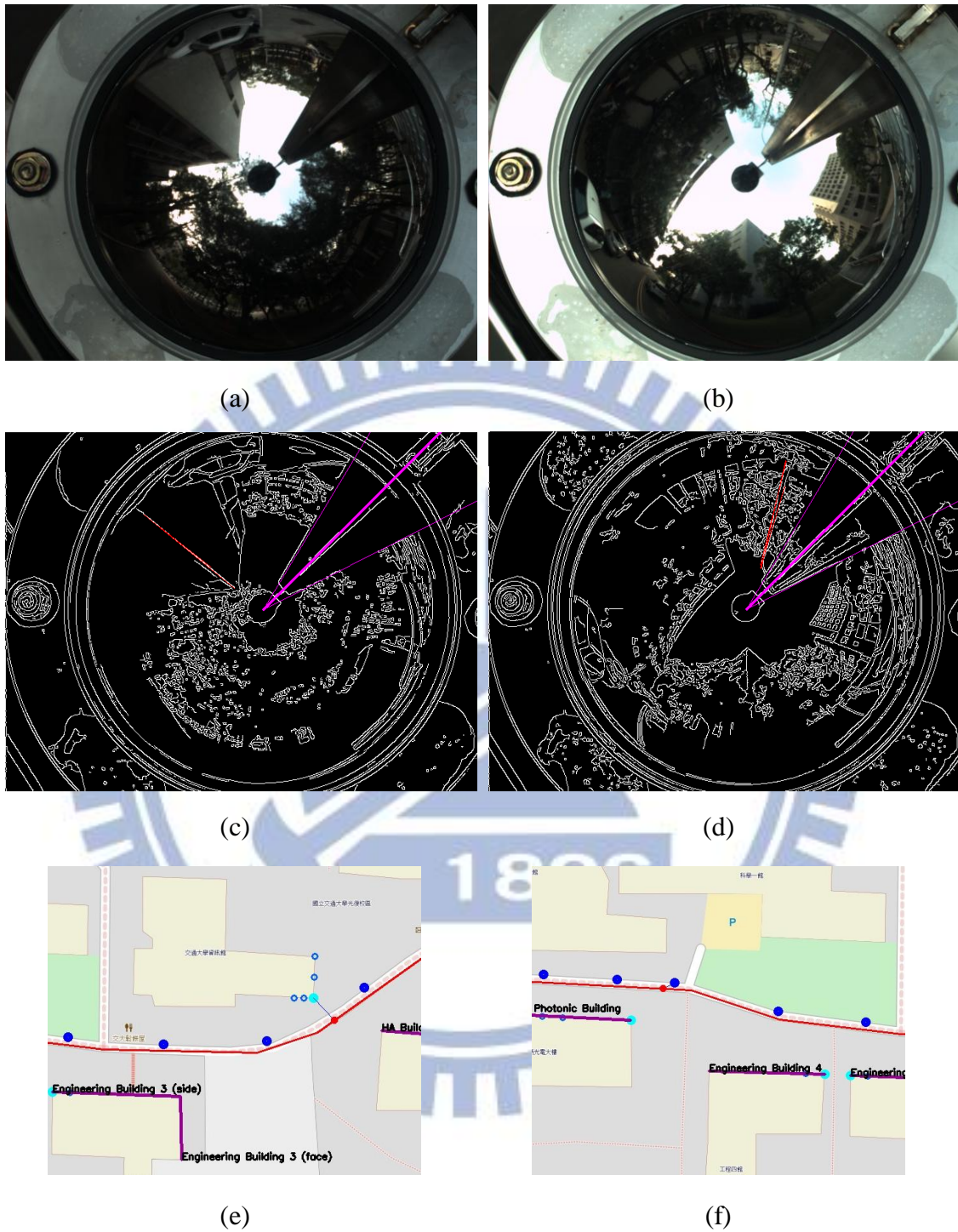


Figure 7.3 An experimental result of detecting the line features and locating the vehicle. (a) An omni-image acquired from the omni-camera. (b) Another omni-image acquired from the omni-camera. (c) A line feature detected by the system. (d) Another line feature detected by the system. (e) The location of the vehicle computed by the system and indicated by the red point. (f) Another location of the vehicle computed by the system and indicated by the red point.

After locating the vehicle, the system can use the result to show the AR image at the mobile device. Figure 7.4 shows an example of such results for tour guidance. We show images of the vehicle on the path in Figures 7.4 (a) and (d). Then, the system augmented the building name in the passenger-view image on the mobile device as shown in Figures 7.4 (c) and (f).

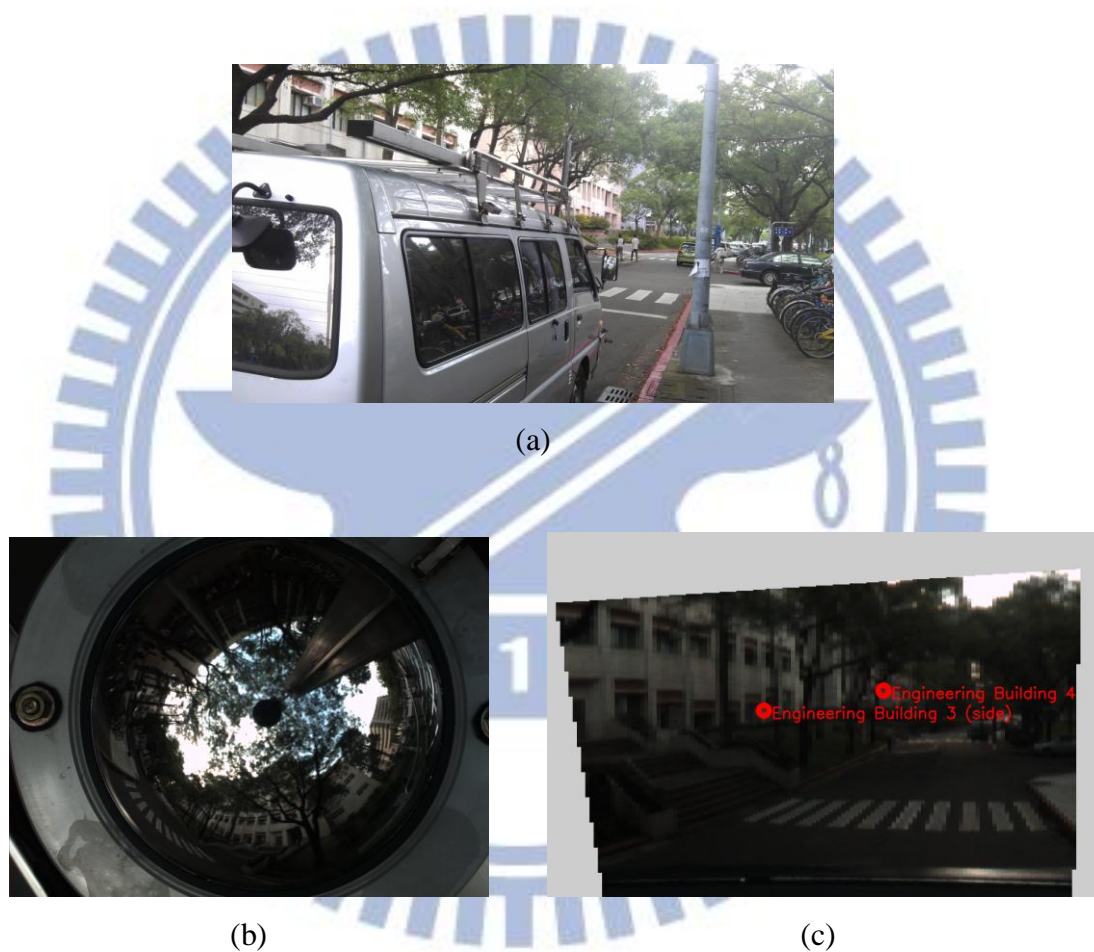


Figure 7.4 AR-based navigation. (a) An image of the vehicle on the path. (b) An omni-image acquired with the omni-camera. (c) The passenger-view image with an augmented building name. (d) Another image of the vehicle on the path. (e) Another omni-image acquired with the omni-camera. (f) Another passenger-view image augmented with the building name.



(d)



(e)



(f)

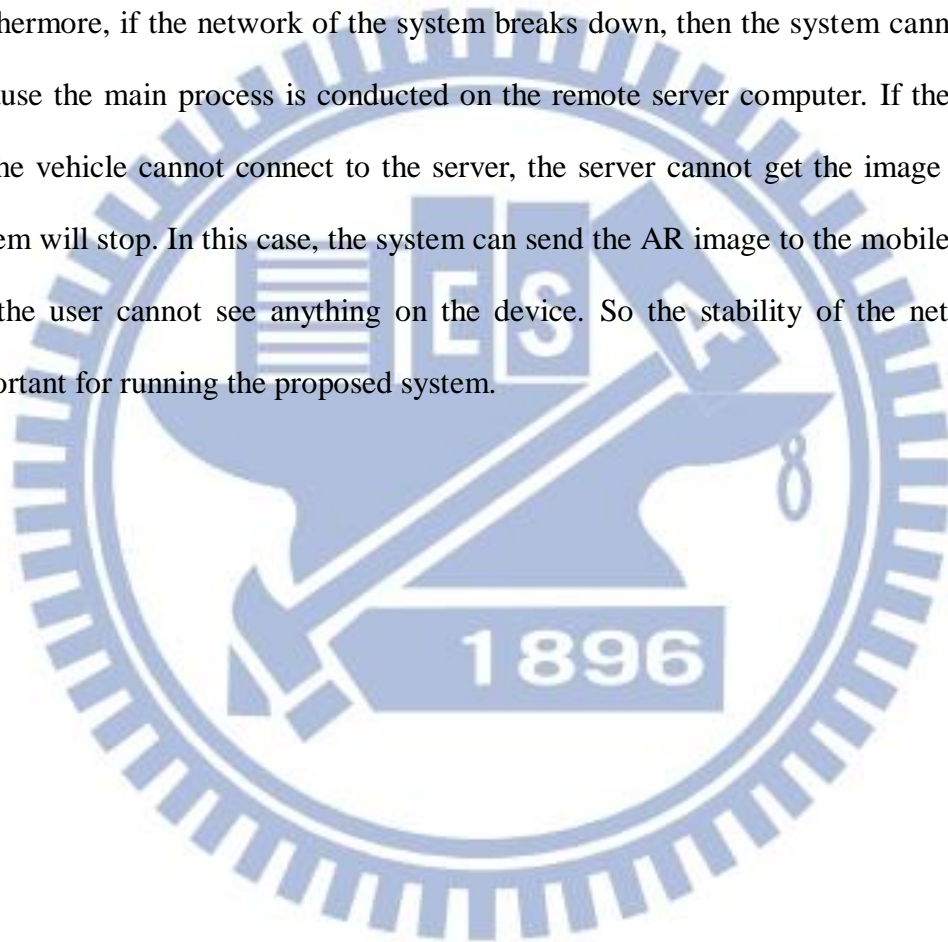
Figure 7.5 AR-based navigation. (a) An image of the vehicle on the path. (b) An omni-image acquired with the omni-camera. (c) The passenger-view image with an augmented building name. (d) Another image of the vehicle on the path. (e) Another omni-image acquired with the omni-camera. (f) Another passenger-view image augmented with the building name (continued).

7.2 Discussions

The experimental results of the proposed tour guidance system presented previously show that it is feasible to use the omni-camera equipped on the vehicle to detect the vertical line features for locating the vehicle by using the vertical-line features. Then, it is also practical to generate the passenger-view image and augment the building name on it for display on user's mobile-device screen.

However, the proposed system still has some problems. If the features in the

environment are too few for the system to detect, the system might have errors because erroneous localization of the vehicle. In more detail, if the distance between two features is too large, then the system will use the estimated vehicle speed to locate the vehicle for a long distance. The error of the location will possibly accumulate to a very large amount and make the system show wrong AR information on the user's mobile-device screen. Therefore, the path we choose is important for the system. Furthermore, if the network of the system breaks down, then the system cannot work because the main process is conducted on the remote server computer. If the system on the vehicle cannot connect to the server, the server cannot get the image and the system will stop. In this case, the system can send the AR image to the mobile device, but the user cannot see anything on the device. So the stability of the network is important for running the proposed system.



Chapter 8

Conclusions and Suggestions for Future Works

8.1 Conclusions

A tour guidance system by augmenting reality techniques for uses in outdoor environments by using an omni-camera imaging device has been proposed. To design such a system, several techniques have been proposed as summarized in the following.

1. *A method for learning the environment map* has been proposed, which generates the environment map for the system to locate the vehicle position and compute the position of the building.
2. *A method for detecting vertical lines in omni-images* has been proposed, which is based on the canny edge detector and detects vertical lines, complete or broken, with widths.
3. *A method for locating the vehicle along the path* has been proposed, which uses the result of vertical-line detection to conduct vertical-line matching by the LCS algorithm to provide the position of the vehicle on a map.
4. *A method for generating the passenger-view image* has been proposed, which generates, as the base of the AR image, the view of the front passenger on the user's mobile-device screen, not seen by the user himself/herself but also other passengers in/outside the car with hand-held mobile devices with wireless communication capabilities.

5. *A method for tour guidance in the park area* has been proposed, by which the user can see augmented passenger-view images with building names on them on the user's mobile-device screen.

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

8.2 Suggestions for Future Works

According to the experience obtained this study, in the following we make suggestions of some interesting issues, which are worth further investigation in the future.

1. Increasing the speed of computations in feature detection and vehicle localization for realtime applications.
2. Developing the capability of detecting features of different shapes adapt the proposed system to more diversified environments.
3. Developing more applications of the proposed augmented reality techniques using the omni-camera system and the vehicle.
4. Adding the capability of detecting various features acquired by the omni-cameras on a fast-moving vehicle.
5. Including more useful information into the environment map for vehicle localization, such as stores, vendors, lakes, etc.

References

- [1] B. C. Chen and W. H. Tsai, "A Study on Tour Guidance by Car Driving in Park Areas Using Augmented Reality and Omni-vision Techniques," in *Computer Vision, Graphics, and Image Processing*, Aug 2012.
- [2] Gandhi and M. M. Trivedi, "Motion analysis for event detection and tracking with a mobile omni-directional camera," *ACM Multimedia Systems Journal, Special Issue on Video Surveillance*, vol. 10, no. 2, pp. 131–143, 2004.
- [3] P. H. Yuan, K. F. Yang, and W. H. Tsai, "A Study on Monitoring of Nearby Objects around a Video Surveillance Car with a Pair of Two-camera Omni-directional Imaging Devices," *Proceedings of 2010 International Computer Symposium (ICS)*, National Chiao Tung University, Hsinchu, Taiwan, pp. 325-330, Dec. 2010.
- [4] S. W. Jeng and W. H. Tsai, "Using pano-mapping tables for unwarping of omni-images into panoramic and perspective-view images," *Journal of IET Image Processing*, Vol. 1, No. 2, pp. 149-155, June 2007.
- [5] Y. T. Kuo and W. H. Tsai, "A new 3D imaging system using a portable two-camera omni-imaging device for construction and browsing of human-reachable environments," *Proceedings of 2011 International Symposium on Visual Computing*, pp. 484-495, Las Vegas, Nevada, USA.
- [6] M. Betke and L. Gurvits, "Mobile robot localization using landmarks," *IEEE Transactions on Robotics and Automation*, Vol. 13, No.2, pp. 251-263, April 1997.
- [7] C. T. Ho and L. H. Chen, "A high-speed algorithm for elliptical object detection," *IEEE Transactions on Image Processing*, Vol. 5, No. 3, pp. 547-550,

March 1996.

- [8] C. J. Wu, “New Localization and Image Adjustment Techniques Using Omni-Cameras for Autonomous Vehicle Applications,” *Ph. D. Dissertation*, Institute of Computer Science and Engineering, National Chiao Tung University, Hsinchu, Taiwan, Republic of China, July 2009.
- [9] T. Grosch, “PanoAR: Interactive Augmentation of Omni—Directional Images with Consistent Lighting,” *Proc. Computer Vision Computer Graphics Collaboration Techniques and Applications (Mirage '05)*, University of Koblenz-Landau, Germany, pp. 25-34, 2005.
- [10] Lee, J.W., You, S., Neumann, U, “Tracking with Omni-Directional Vision for Outdoor AR Systems,” *Proceedings of IEEE ACM Int'l Symposium on Mixed and Augmented Reality (ISMAR 2002)*, Darrnstadt, Gkrmany, October 2002.
- [11] G. Reitmayr and T.W. Drummond, “Going out: Robust model based tracking for outdoor augmented reality,” *Proc. IEEE Int'l Symp. Mixed and Augmented Reality (ISMAR)*, Santa Barbara, California, USA, pp. 109–118, 2006.
- [12] M. Tonnis, C. Sandor, G. Klinker, C. Lange, and H. Bubb. “Experimen-tal evaluation of an augmented reality visualization for directing a car driver’s attention.” *Proc. of IEEE and ACM International Symposium on Mixed and Augmented Reality*, pp. 56–59, Vienna, Austria, Oct. 2005.
- [13] B. D. Lucas and T. Kanade, “An iterative image registration technique with an application to stereo vision,” *Proceedings of 7th International Joint Conference on Artificial Intelligence*, Vancouver, Canada, pp. 674–679, 1981.
- [14] T. Mashita, Y. Iwai, and M. Yachida, “Calibration method for misaligned catadioptric camera,” *IEICE Transactions on Information & Systems*, vol. E89-D, no. 7, pp. 1984-1993, July 2006.
- [15] H. Ukida, N. Yamato, Y. Tanimoto, T. Sano, and H. Yamamoto,

“Omni-directional 3D Measurement by Hyperbolic Mirror Cameras and Pattern Projection,” *Proceedings of 2008 IEEE Conference on Instrumentation & Measurement Technology*, Victoria, BC, Canada, May 12-15, 2008, pp. 365-370.

