國立交通大學

多媒體工程研究所

碩士論文

以俯視式環場電腦視覺及行動裝置作擴增實境式室內導覽

A Study on Indoor Navigation by Augmented Reality and Down-looking Omni-vision Techniques Using Mobile Devices

研究生：謝孟原

指導教授：蔡文祥　教授

中華民國一百零一年六月

以俯視式環場電腦視覺及行動裝置作擴增實境式室內導覽

A Study on Indoor Navigation by Augmented Reality and Down-looking Omni-vision Techniques Using Mobile Devices

研 究 生：謝孟原　　Student：Meng-Yuan Hsieh

指導教授：蔡文祥　　Advisor：Wen-Hsiang Tsai

國 立 交 通 大 學

多 媒 體 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Multimedia Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

June 2012

Hsinchu, Taiwan, Republic of China

中華民國一百零一年六月

以俯視式環場電腦視覺及行動裝置作擴增實境式室內導覽

研究生：謝孟原　　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

本論文提出了一個結合電腦視覺及擴增實境技術在行動裝置上使用的室內導覽系統。此系統以在室內環境天花板上安裝的魚眼攝影機作為基礎硬體架構。在人物定位方面，提出了一個以電腦視覺為基礎的方法，藉由分析魚眼影像來偵測使用者的活動資訊。為了得到影像中人物的真實空間位置，我們也提出了一個空間映射的方法，來進行影像座標與真實空間座標的轉換。此外我們也整合了三項技術來進行人物方向的偵測，分別為(一)分析使用者的移動路徑、(二)利用行動裝置上的方向感測器、以及(三)藉由行動裝置上所貼一長條色彩標記，在魚眼影像中分析該標記來進行方向偵測。另亦提出一適用於室內路徑的規劃方法，藉由分析建築平面圖來得到障礙物區域，並以此為基礎得到障礙物迴避方向來進行路徑規劃。伺服器會將導覽資訊傳送至行動裝置上的使用者端，此資訊包括了定位資訊、周遭環境地點及導覽路徑。使用者端接收到的導覽資訊會被覆蓋在行動裝置影像中對應的真實物件上，來提供擴增實境導覽介面。此外本研究也提出了一個方法來估測行動裝置上攝影機的可視角，並以此建立一個轉換矩陣來將真實空間中的點轉換到影像平面上。最後，實驗結果也顯示出了本研究所提出方法的可行性。同時，定位資訊的精確測量結果也顯示了此系統在提供精確導覽資訊的能力。

A Study on Indoor Navigation by Augmented Reality and Down-looking Omni-vision Techniques Using Mobile Devices

Student: Meng-Yuan Hsieh Advisor: Wen-Hsiang Tsai

Institute of Multimedia Engineering, College of Computer Science
National Chiao Tung University

ABSTRACT

When people visit new indoor places or complicated indoor environments, there usually needs a navigation system to guide them to desired destinations. In this study, an indoor navigation system based on augmented reality (AR) and computer vision techniques by the use of a mobile device like an HTC Flyer or an iPad is proposed.

At first, an indoor vision infra-structure is set up by attaching fisheye cameras on the ceiling of the navigation environment. The user's location and orientation are detected at a server-side system, and the analysis results are sent to the client-side system. Furthermore, the server-side system also sends the surrounding environment information and the navigation path to the client-side system, which runs on the user's mobile device. The client-side system then displays the information in an AR way, which provides clear information for a user to conduct the navigation.

For human localization, a vision-based localization technique is proposed, which analyzes images captured from the fisheye cameras, and detects human activities in the environment. In order to transform coordinates of image points into the real-world space, a space-mapping technique is proposed. Furthermore, three techniques are

integrated together to conduct human orientation detection effectively. The first is analysis of human motions in consecutive images. The second is utilization of the orientation sensor on the user's mobile device. The last is localization of the color edge mark attached on the top of the mobile device using omni-images. These techniques are integrated together to provide a reliable human orientation detection system.

A path planning technique for use to generate a path from a spot to a selected destination via the use of an environment map is also proposed. The environment map is constructed from a floor plan drawing of the indoor environment. An obstacle avoidance map is created from the floor plan drawing, which is used to determine the avoidance direction when a path collides with an obstacle in the environment.

Finally, the navigation information is overlaid onto the image shown on the mobile device to provide an AR navigation interface. A method for estimation of the field-of-view of the camera on the mobile device is proposed. The field-of-view is used to construct a transformation matrix, by which real-world points can be transformed into the screen plane, so that the navigation information can be overlaid onto the corresponding real-world objects in the images to accomplish the AR function of the system.

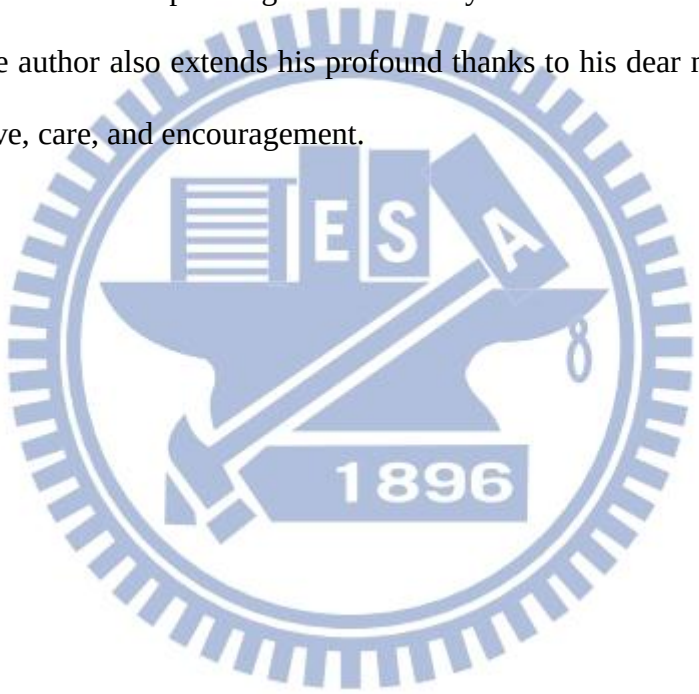
Good experimental results are also presented to show the feasibility of the proposed methods for real applications. Precision measures and statistics showing the system's effectiveness in producing precise data for accurate visiting target displays and environment navigations are also included.

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.

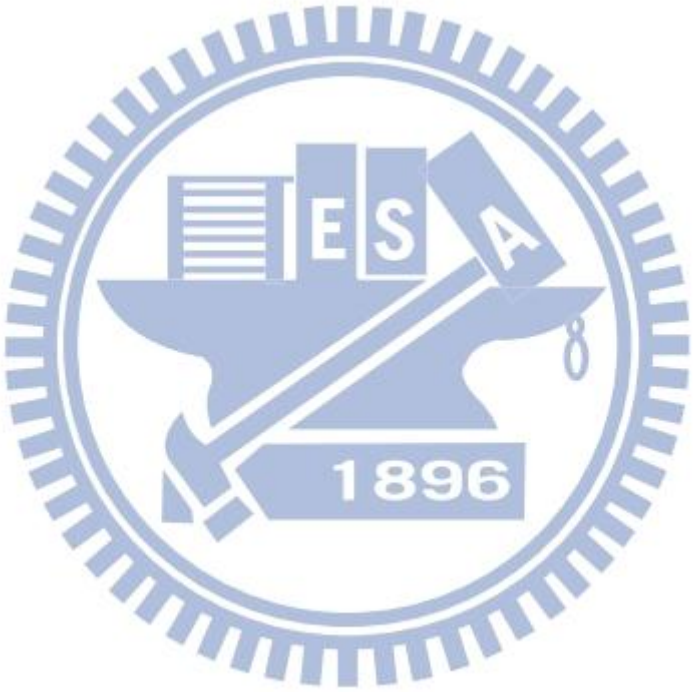
Finally, the author also extends his profound thanks to his dear mom and dad for their lasting love, care, and encouragement.



CONTENTS

LIST OF FIGURES

LIST OF TABLES



Chapter 1
Introduction

1.1 Background and Motivation

When people visit new places or complicated indoor environments, such as company buildings, large labs, malls, department stores, etc., there usually needs a navigation system to guide them to desired destinations. Common navigation systems use the global positioning system (GPS) to retrieve position data, but the GPS is generally not suitable for use to acquire indoor locations, since signals will be attenuated and scattered by roofs, walls, and other objects in indoor environments, resulting in imprecise localization readings. In this study, it is desired to design an indoor navigation system using a different localization technique. Specifically, we try to design a vision-based localization technique to analyze the images captured from fisheye cameras installed on ceilings in indoor environments and detect human activities in the environments.

Meanwhile, we try to use mobile devices as user-end devices. Mobile devices are getting more and more popular nowadays and are used more and more widely in various applications. In recent years, many mobile devices become commercially available, such as smart phones and tablets equipped with more advanced function units like high-speed CPUs, graphics processing units (GPUs), digital cameras, device orientation sensors, etc. Therefore, application developers can design many complicated mobile applications or services that assist people in real-life events due to the high-speed computational and advanced capabilities of the devices.

Moreover, as the on-device camera getting cheaper and more common, we can use them to develop more interesting and useful applications by combining real-world images captured from cameras with virtual augmentations created by computers. In other words, the real-world environment can be augmented by computer-generated objects to enhance the perception of the real world, and this is the so-called *augmented reality* (AR) technique. In this study, we try to design an indoor navigation system by the AR technique using mobile devices. We want to overlay artificial navigation instructions mentioned above onto the real images captured with the camera in real-time, so that users can just take their mobile devices and conduct indoor-environment navigations conveniently. The concept of the proposed system is shown in Figure 1.1.

In summary, the goal of this study is to develop an indoor navigation system with the following capabilities.

1. Working in indoor environments, and being able to detect users' positions and orientations.
2. Integrating real images with virtual augmentations, such as the current position, the next moving direction to the desired destination, nearby visiting target information, etc., to provide users convenient and clear navigation interfaces.
3. Planning a proper path from a user's location to a desired destination, and updating the path dynamically when the user moves to a location not in the path.

1.2 Review of Related Works

In this section, we conduct a survey of works about indoor navigation and related techniques, such as human localization, human orientation detection, navigation path planning, and AR techniques.

Figure 1.1 Concept of proposed indoor navigation system using augmented reality technique

1.2.1 Review of Related Indoor Navigation Works

In recent years, with outdoor navigation systems become more popular and widely used, there are more and more researches about indoor navigation trying to satisfy the demands for indoor environments. Lukianto, et al. [1] proposed an indoor navigation system for use on the smart phone, which is based on an inertial navigation system (INS) and provides the position, speed, and orientation of the user. Ozdenizci, et al. [2] proposed a near field communication (NFC) based system, which detects the user's position by touching NFC tags with a smart phone.

Besides the sensor-based navigation systems mentioned above, several systems using image processing techniques have been proposed. The most common technique used in image-based systems is *marker-based navigation* with camera phones [3, 4], in which a user must point the phone's camera at a marker, and the system then will recognize it and know where he/she is located.

Many other systems also use image-based techniques for AR. Werner et al. [5] proposed a method to detect human positions in indoor environments by a combination of image processing systems with a distance estimation algorithm using

the camera of a mobile device. Hile and Borriello [6] developed an indoor navigation system that can find the camera pose by detecting the landmark in the phone camera image and matching it with previously-cached landmarks, and then overlaying the information onto the images.

1.2.2 Review of Related Augmented Reality Works

Augmented Reality enhances the real world with virtual objects or digital information, so it has been used in many fields. For example, it can be used to help mechanics to perform maintenance and repair tasks [7], treatment for psychological disorders [8], context visualization [9], etc.

We develop our navigation system by the AR technique, and there are also other systems using AR techniques. Jongbae and Heesung [10] proposed a vision-based indoor navigation system, which recognizes the location of users by marker detection and image sequence matching on images captured from a wearable camera, and display navigation information in the AR way. Miyashita, et al. [11] designed a museum guide system, which uses a markerless tracking technique and an AR platform — Unifeye SDK.

1.2.3 Review of Related Human Localization Works

The human localization techniques of navigation systems mentioned in the previous sections can be roughly classified into the two types of sensor-based and image-based. Sensor-based localization techniques usually need infrastructures with infrared, RFID, NFC tags [2], or other customer-designed hardware [1]. Image-based localization techniques usually use markers attached on the environments or the features acquired from the images captured by cameras [5] [6].

In this study, we propose an image-based localization technique, which uses

fisheye cameras to capture images and detect the user's location. The fisheye camera has the advantage of possessing wider fields-of-view, so we can use it to observe wider regions in many applications. In our system, 2-D image points must be transformed into the 3-D global space to get the actual position of the user. For this purpose, we tried to use a space-mapping method proposed in [12]. But this method is based on the use of fixed cameras. When the positions or other configurations of the cameras are changed, we have to redo the works again. In this study, we propose a method to solve this problem and improve the above method for more flexibility and better usability.

1.2.4 Review of Related Path Planning Works

When a navigation system gets a user's location and the user wants to reach a certain destination, then the system should plan a path from the user position to the destination.

About the path planning technique, Borenstein and Koren [13] proposed a real-time collision avoidance method using a technique named Vector Field Histogram, which can detect unknown obstacles and avoid collisions. Hwang, et al. [14] proposed a path planning method by a path graph optimization technique which triangulates the world space into a mesh representation, and then extract an optimized path graph from the mesh. Bruce and Veloso [15] proposed a path planning technique named rapidly-exploring random trees (RRTs) by waypoint caching and adaptive cost penalty search, which improve re-planning efficiency and the quality of generated paths.

1.3 Overview of Proposed Methods

The most important part of every navigation system is the localization function. As discussed previously, we usually need a GPS to retrieve the position data in an outdoor environment, but the GPS is not suitable for indoor environments due to its rough localization precision. Therefore, we propose a new method for indoor localization in this study under the assumption that there is only one user in the indoor environment taken care of by the system. At first, we have to build a top-down vision infrastructure in the indoor environment, which has a sufficient number of fisheye cameras installed on the ceiling. Then, an environment map model is created, which includes the location information of the cameras and the visiting targets for guidance. Each *visiting target* means a place or object in the real environment, such as an exit, a restroom, a water dispenser, and others that people might be interested in, and this term will be used in the subsequent sections. The cameras are with fisheye lenses which have wide fields-of-view. Such cameras can be deployed to monitor the entire environment with a smaller number of them. Then, we analyze the images captured from the cameras to detect the user's position on a server-side system.

After getting the user's position, his/her orientation must be detected to decide what the camera on the user's mobile device "sees." Then, the system can send relevant navigation information to the client-side system through a wireless network in the environment, and the user at the client site can realize what the visiting targets in sight on the device display screen are or how to get to its desired destination. In our system, we detect the user's orientation by integrating three different techniques, which respectively are human motion analysis, localization by the e-compass data, and detection of a color edge mark on top of the hand-held client device at the client site.

In order to guide a user to a desired destination, we propose a path planning technique for indoor navigation. The path planning technique is based on a floor plan of the indoor environment in the form of a graphic picture. By the technique, we analyze the floor plan to localize obstacle and walkable regions in the plan, and use the resulting information to plan paths according to the destination which is taken as input to the technique.

After getting the user position and orientation, the server system sends the navigation information to the client system, which then displays the information on the device screen at the client site. The navigation information data includes the name and distance of visiting targets in sight of the user, the navigation path to the desired destination, etc. The client system will map the information from the real world to the screen of the hand-held device. Then, the visiting target information or the navigation path can be overlaid onto the real places or objects shown in the current image taken of the environment by the built-in camera of the device. In other words, the device displays the navigation information *in an AR way*. As such, the user can understand the surrounding environment easily and intuitively.

1.4 Contributions

The major contributions of this study are listed in the following.

1. A new AR-based indoor navigation system using computer vision is proposed to satisfy the demands of guidance or browsing of indoor environments.
2. An image-based localization method by analyzing the images captured from the fish-eye cameras affixed on the ceiling is proposed to compensate for the insufficiency of the GPS in the indoor environment.
3. An augmented reality interface is proposed to provide a user with surrounding

navigation information and the navigation path from the position of the user to the specified destination.

1.5 Thesis Organization

The remainder of this thesis is organized as follows. In Chapter 2, we introduce the configuration of the proposed system and the system process in detail. In Chapter 3, we introduce the proposed process for learning of an indoor environment, which includes the data that we will use in the proposed system. In Chapter 4, the proposed user localization method for indoor environments and the proposed user orientation detection method are described. In Chapter 5, we introduce the proposed path planning technique. In Chapter 6, we describe the proposed AR technique, a method to conduct the perspective transformation for information displays on the user's device, and the adopted technique for rendering augmentations on real images. In Chapter 7, some experimental results to show the feasibility of the proposed techniques for indoor navigation are presented. At last, conclusions and some suggestions for future works are given in Chapter 8.

Chapter 2
Ideas of Proposed Methods and System Design

2.1 Ideas of Proposed Method

We propose an image-based localization technique for AR-based guidance of indoor environments in this study. The system analyzes the omni-images captured from the cameras affixed on the ceiling, and then finds the user's foot points in the omni-images. When we get the user's foot points, we transform their coordinates in the *image coordinate system* (ICS) into the *global coordinate system* (GCS) to get the actual position of the user in the indoor environment. In order to conduct the above transformation, we construct a mapping table between the ICS and the GCS in advance.

Next, we must detect the user's orientation after detecting the user's location. In order to accomplish this aim, the simplest way is to track the user's locations in consecutively acquired images, and use the resulting motion vectors of the user's foot points to compute the user's orientation. But when the user is not walking, this method will not work because there is then no more moving vector for use. In this situation, we propose other techniques to overcome the problem. The first technique is to utilize the orientation sensor installed in the user's device mentioned previously. The orientation sensor measures the azimuth angle of the device by detecting changes and disturbances in the magnetic field in the surrounding environment. However, according to our experimental experience, the azimuth values detected are not stable

enough for our application due to indoor magnetic interferences from various sources. Therefore, we propose a second technique to improve the stability of detected orientations, that is, to attach a *color edge mark* on the top edge of the user device, and detect this line mark appearing in the omni-image to compute a more accurate orientation of the user at each visiting target.

In addition, in order to guide users to their desired destinations, we propose further a path planning technique for the proposed indoor AR navigation system. An environment map model is constructed first from a graphic drawing of the floor plan of the environment. Then, walkable regions in the floor plan are detected by image processing techniques with the graphic drawing as input. In this way, we can know where the obstacles are in the environment. The orientations of the obstacles then are analyzed to decide how to avoid them and where to go next. When a user wants to go to a destination, the system will search the constructed environment map, and get the destination point in the map. In the meantime, the system will plan a path starting from the user position and ending at the destination. When the planned path collides with any obstacle, it follows the orientation of the obstacle's boundary to avoid the collision and go to the next immediate visiting spot. Repeating the above steps until reaching the appointed destination, we can get a complete navigation path finally as the desired path planning result.

When the client-side system receives the *navigation information* sent from the server, the system will display the information on the device screen. The navigation information includes the *visiting target information* and the navigation path itself. The visiting target information includes the name of the visiting target and its coordinates in the GCS. The navigation path contains the GCS coordinates of the points on the path. In order to display the information in an AR manner, the client-side system must transform the GCS coordinates onto a 2D screen plane. The field-of-view of the

camera of the client device must also be estimated to get a perspective projection matrix. With the matrix, the 3D points of the navigation information can be transformed into the 2D screen plane. Then, the navigation information can be overlaid onto the real places or objects in the image taken of the current scene, and the user can so understand the surrounding environment easily, achieving the major goal of AR-based indoor environment guidance of this study. This step of navigation information overlaying on real environment images for displays on the user's mobile device will be called *display rendering* in the sequel of this thesis.

2.2 Ideas of System Design

In this study, the proposed system is of a client-server architecture, which may be decomposed into two parts: a server side and a client side. The server-side system is used for conducting complicated works with heavy computations, and it runs on a centralized computer. The server-side system will be introduced in more detail in Section 2.2.1. The client-side system runs on the user's mobile device, which obtains navigation information from the server-side system and displays it on the screen of the device. The client-side system will be introduced in more detail in Section 2.2.2. Finally, the cooperation between the client and server sides will be introduced in Section 2.2.3.

2.2.1 Server-side System

The server-side system runs on a centralized computer as mentioned, and is connected to the cameras on the ceiling through a local area network. In the learning stage, we build an environment map, which includes environment information such as target locations, target titles, and camera locations. In the navigation stage, the server

accesses the omni-images captured from the cameras, and analyzes the omni-images to detect the user's location and orientation at each visited spot. After the server detects a user via images acquired by the cameras, it sends the user's location, orientation, and the information of nearby visiting targets to the user's client-side system. All of such information will be updated when the user moves. When the user wants to reach a certain destination, the server will receive a request from the client, and then plan a path from the user's location to the destination, and send a set of intermediate points of the path to the user's client-side system to display.

As a whole, the server is designed mainly for conducting human localization and path planning, and these two tasks are both heavy computational works. Because the client-side system runs on the user's mobile device, which has lower power and inferior computational capabilities than the centralized computer, conducting these heavy computational works on the server can increase the computational performance and reduce the battery power usage of the client-side system.

2.2.2 Client-side System

The client-side system runs on the user's mobile device. Because the mobile device held by the user (like an iPad) has lower power and inferior computational capabilities than a laptop or desktop computer, the client-side system on it must be assigned as few works as possible to reduce the power consumption and increase the computational performance. Therefore, most tasks carried out by the client-side system are limited to be those related to information displays, such as view projection, display rendering, and creation of the navigation path's geometric shape (arrows, thick line segments, etc).

When a user enters the environment, the user's client-side system is connected to the server through a network and receives relevant information from the server. Then,

the client-side system just needs to display the information on the screen of the user's mobile device.

2.2.3 Cooperation between Client and Server Sides

The server and client side systems are described in Section 2.2.1 and Section 2.2.2. Here we describe the cooperation between the client-side and server-side systems in more detail. An illustration of the cooperation between the two systems is shown in Figure 2.1.

When the client is connected to the server, the latter will begin to detect the user's location and orientation, and send the location coordinates, the orientation vector, and the nearby environment information to the user. The information will be updated continuously to make sure that the user can receive correct and immediate messages. When the user wants to reach a certain destination, the client-side system will send a request, which includes the name of the destination, to the server. After server receives the request, it will plan a path starting from the user's location and ending at the destination. Finally, a set of intermediate points of the path will be sent to the client.

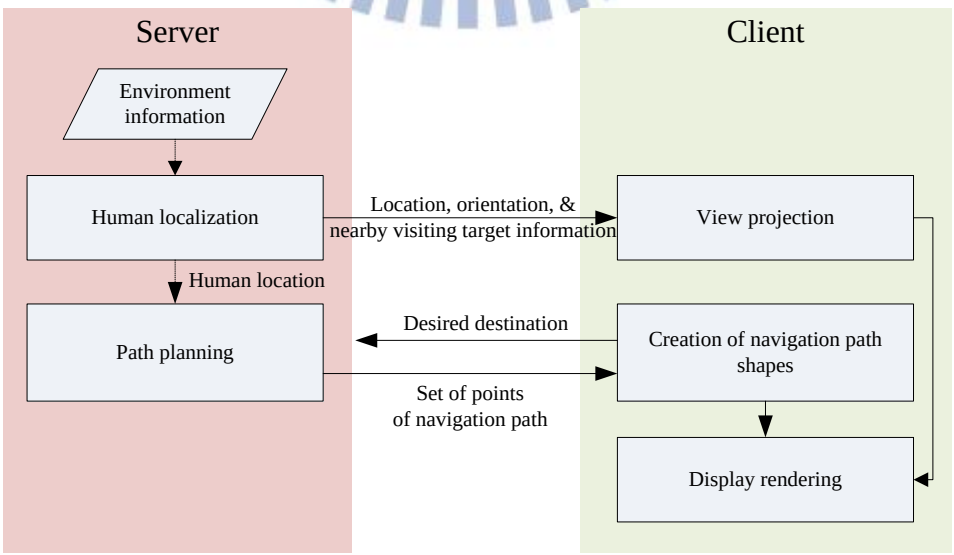


Figure 2.1 Cooperation between client and server sides.

2.3 System Configuration

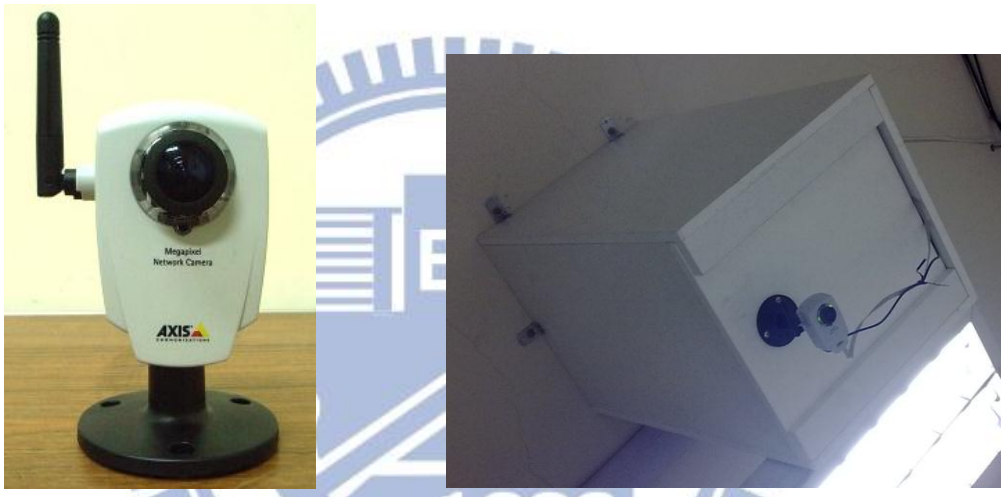
In this section, we will introduce the configuration of the proposed system. The hardware of the proposed system includes fisheye cameras which we use for human detection, and the mobile device which we use as the client-side device. It will be introduced in more detail in Section 2.3.1. In Section 2.3.2, we will describe how to connect the hardware over the network, and how it operates. Finally, we will introduce the software development environment and the operating system we use both in the server-side system and in the client-side system.

2.3.1 Hardware Configuration

The camera we use in this study is of the model of Axis 207MW, which is made by Axis Communications, and the original lens is replaced with a fisheye lens in this study to expand its field-of-view. The Axis 207MW camera has a dimension of 85×55×40mm (3.3”×2.2”×1.6”, not including the antenna), and a weight of 190g (0.42 lb., not including the power supply). Its appearance is shown in Figure 2.2(a). The maximum resolution of the images captured with it is up to 1280×1024 pixels. For performance efficiency, we use the resolution of 640×480 pixels in our system, and the frame rate is up to 15 fps. The cameras can be accessed through wireless networks (IEEE 802.11g/b), but for speed improvement, we access the cameras through the Ethernet.

We build our experimental environment in the Computer Vision Lab at National Chiao Tung University by installing several fisheye cameras on the ceiling of the lab. (see Figure 2.2(b)). The images captured from the cameras are analyzed by the centralized computer to detect the user’s location and orientation. The server sends the navigation information to the users’ mobile device so that the user can begin the

navigation. The mobile device we use in the experiment is a HTC Flyer tablet made by HTC Corporation. Its appearance is shown in Figure 2.3. The HTC Flyer has a dimension of 195×122×13.2mm (7.7”×4.8”×0.5”) and a weight of 420g (0.93 lb). It has a screen size of 7 inches, a camera acquiring 5-megapixel images, and an e-compass that can detect the device orientation in a magnetic field, etc. The user uses the HTC Flyer as the client device, and connects it to the server through a wireless network.



(a) (b)

Figure 2.2 The camera used in the proposed system. (a) The appearance of the camera. (b) The camera installed on the ceiling in the indoor environment.

Figure 2.3 The HTC flyer used as the client device in this study.

2.3.2 Network Configuration

Using the Ethernet is more reliable for our application in this study than using a wireless network. Therefore, the cameras and the centralized computer are connected through a *local area network* (LAN) in this study. The server can access the images captured from the cameras in a more reliable way through the Ethernet, and so one can make sure that the system always accesses correct and immediate images and messages.

The client device we use is a mobile device, so it must access the server through the wireless network. The most commonly-used wireless networks currently are the Wi-Fi and 3G networks, and the client device can access the server and receive the navigation information using both of them. For reliability and speed considerations, we set up a Wi-Fi access point in our experimental environment, and the user can connect to the server through the Wi-Fi network in the environment. A complete network architecture is shown in Figure 2.4.

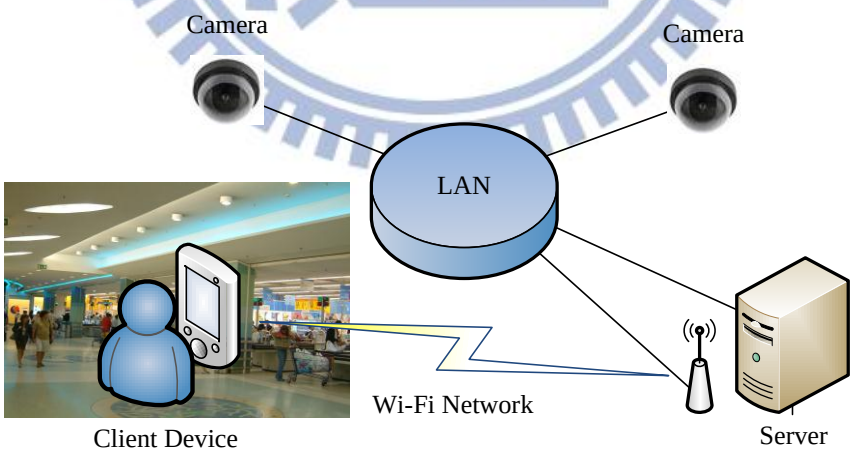


Figure 2.4 The network architecture of the proposed system.

2.3.3 Software Configuration

The server-side system is written in C# programming language using the

Microsoft Visual Studio 2010 development environment, and the system operates on the Windows 7 operating system. The server-side system accesses the cameras by the *AXIS Media Control SDK (AMC SDK),* which is provided by the manufacturer of the cameras, Axis Communications. The AMC SDK provides the application programming interface (API) for developers to access the camera images or control the cameras using C# and C++ programming languages.

As to the client-side system, it is written in the Java programming language and operates on the Android 2.3.4 operating system. The client-side system uses the *Qualcomm's Augmented Reality (QCAR)* platform, which provides many useful functions for AR developments on mobile devices. But in our system, we only use the QCAR to handle the capturing of camera images. The rendering of 3D augmented objects is conducted by the Android OpenGL API.

2.4 System Processes

2.4.1 Learning Process

The goal of the learning process of the proposed system is to establish the environment map, which includes information about the visiting targets, cameras, magnetic fields, and obstacle orientations. The entire learning process is shown in Figure 2.5, and more details of it will be described in Chapter 3. Only a brief description of the process is given here.

First, we establish an environment map in the form of a floor plan drawing. The floor plan is drawn at a specific ratio relative to the actual size of the environment. After specifying the ratio, we compute the corresponding size in the unit of pixel. The use of this scaling ratio is necessary for the transformation between the ICS and the GCS. Next, the visiting targets of the environment are specified on the environment

map. Furthermore, we must also specify the *installation information* of the fisheye cameras. The installation information includes the location and height of the cameras, which is necessary for use in computing the transformation between the *image coordinate system* and the *map coordinate system*.

After the environment map is established, the learning processes can be decomposed into two phases: learning for path planning and learning for human localization. Before we perform path planning, the system must know the information of obstacles. The path planning algorithm can determine how to avoid the obstacles in the environment by the obstacle information. Therefore, the goal is to analyze the information of obstacles, which includes obstacle location and obstacle orientation, in the path planning phase. A more detailed description of obstacle analysis will be given in Section 3.3.3.

In the human localization phase, we calibrate the cameras, including the server-side fisheye cameras and the client-side on-device camera, to map the points between different coordinate systems. A more detailed description of the camera calibration process will be described in Section 3.4. Furthermore, the system detects the user's orientation by aid of the e-compass on the client device. The e-compass, as mentioned before, is an orientation sensor that measures the azimuth angle of the device by detecting the changes and disturbances in a magnetic field around the currently-visited spot. However, the magnetic field will be interfered by the structural steel elements in a building, so the magnetic field does not have an identical distribution at every location in the environment. To learn the magnetic field in the visited environment, we establish an azimuth map, which keeps a record of four direction azimuth values for every sample location in the environment map. A more detailed description of the magnetic field learning process will be described in Section 3.3.4.

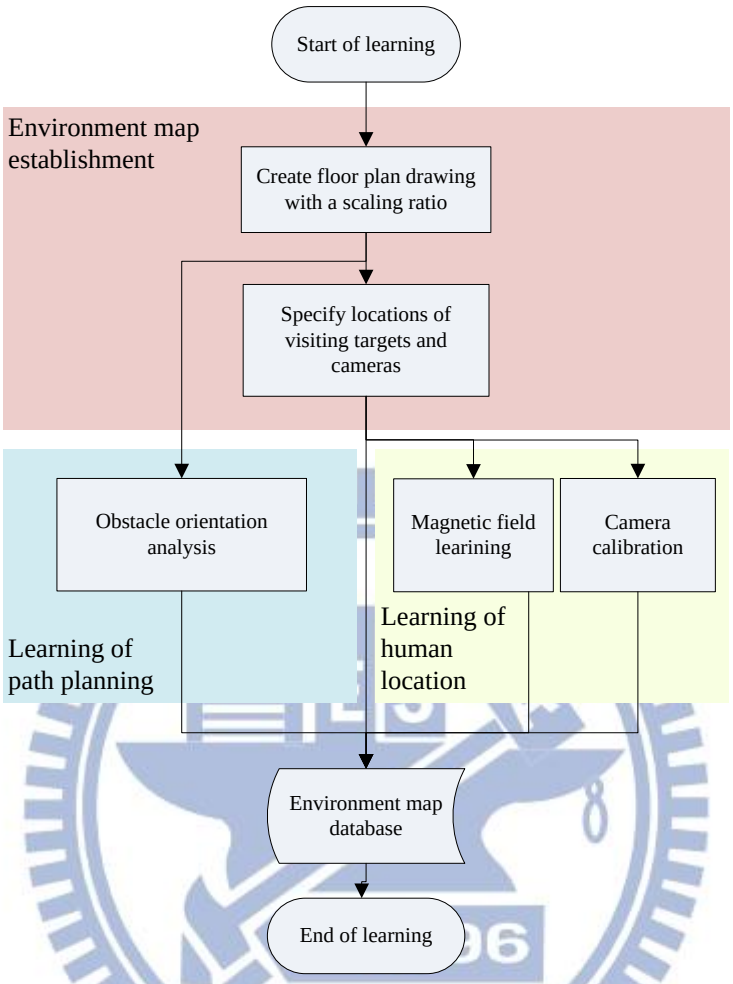


Figure 2.5 Learning process.

After the above learning steps, we have completed the preparation works needed in the navigation stage of the proposed system process. In the next section, we will describe the works conducted in the navigation stage.

2.4.2 Navigation Process

In the navigation stage, the server analyzes the omni-images captured with the cameras continuously, and sends the environment information to the client. The client-side system displays the information on the screen of the user's mobile device. When the user wants to reach a certain destination, the server will plan a path, and

send a set of intermediate points of the path to the client. The entire navigation process proposed in this study is shown in Figure 2.6.

At the server side, the first step is human location detection. The proposed human localization process transforms the detected human location from the ICS into the GCS using the camera information we have acquired in the learning stage. Then the user's location is used in the steps of human tracking and human orientation detection. The objective of the human tracking step is to identify the same human in consecutive video frames, and then compute the user's speed to determine whether the user is walking or not. In the human orientation detection step, we detect the orientation by analyzing the color edge mark, which is on the top of the client device, in the omni-image. However, when the color edge mark is not observable in the omni-image, another technique must be adopted. For this, we compute the orientation by use of detected human motions, or by the azimuth map constructed in the learning stage. Here we also determine the nearby visiting targets seen by the user according to the user's location. Finally, the server sends the information of the user's location and orientation, and the nearby visiting targets to the user's mobile device (the client).

Next, if the server receives a request that the client wants to reach a certain destination, the server begins the path planning process; if not, the server continues to conduct human localization repetitively. At the first step of path planning, the system tries to find a path starting from the user location and ending at its desired destination using the obstacle information analyzed in the learning stage. But the found path may be not of the simplest form; i.e., there may exist two non-connected points in the path that can instead be connected together. In such cases, we simplify the path to be of a simpler form. Finally, a set of resulting intermediate points of the path will be sent to the client.

When the client receives the navigation information mentioned above, it begins

to conduct the work of display rendering by “drawing” the information, which includes the visiting target information and the navigation path, on the device screen for the user to inspect. In order to map real world objects onto the mobile device screen, the first step of the client is to set up a perspective projection by use of the location and orientation of the user. The orientation detected from the server is an azimuth angle, which represents a direction in a horizontal plane. However, a user might tilt the client device to watch the environment at a *pitch angle* rather than at a horizontal angle, so we add the pitch angle value to the detected orientation angle to provide a correct final orientation of the user’s device. The pitch angle can be obtained from the orientation sensor of the client device.

After the client receives a navigation path, it creates a geometric shape of the path; specifically, the client will transform the set of intermediate points of the path into an arrow shape pointing to the destination. Finally, the client begins to draw the information and overlays the generated virtual objects onto the real image taken of the current scene to accomplish the display rendering task.

The above processes of both the server side and client side are run repeatedly until the client terminates the navigation system.

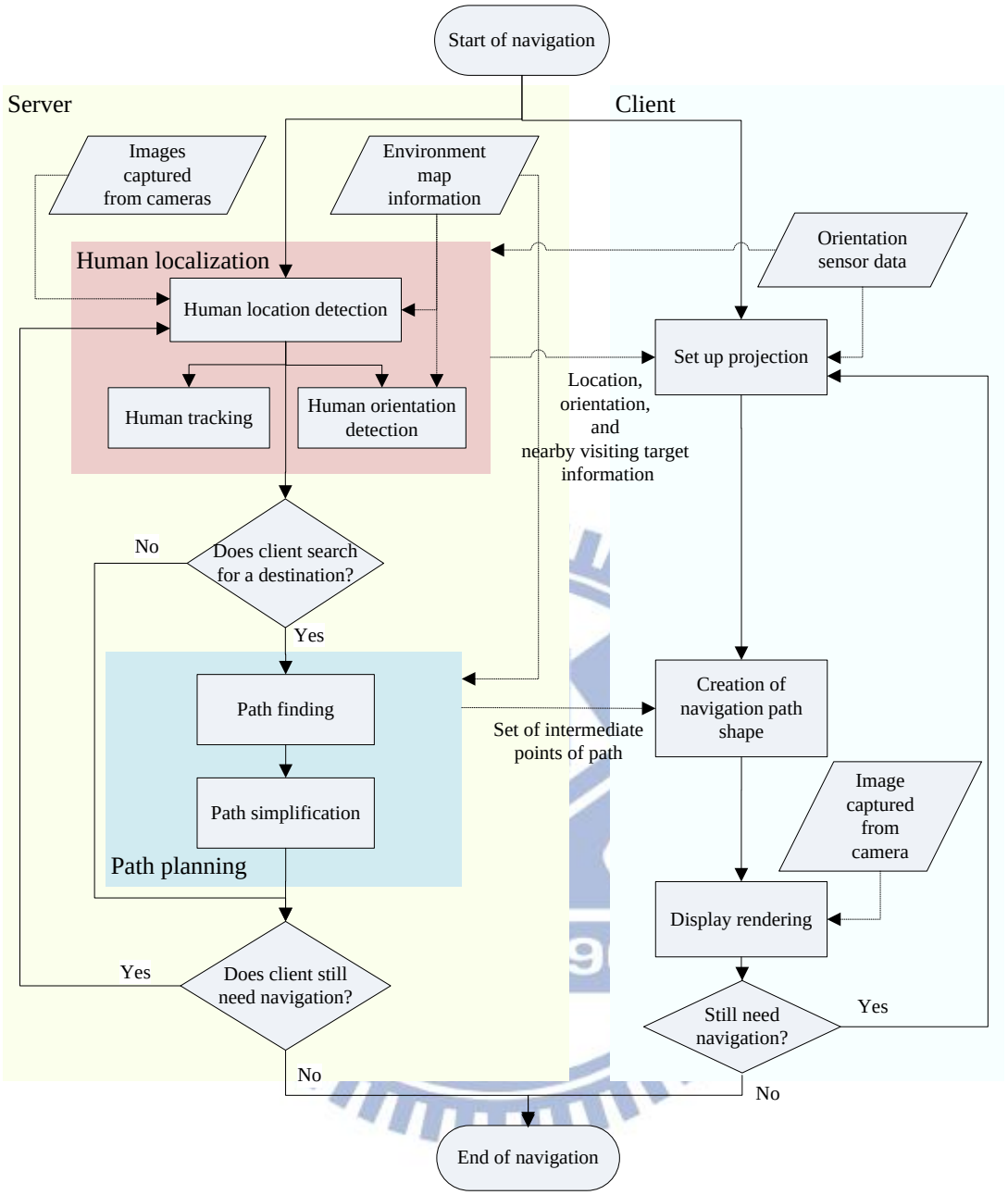


Figure 2.6 Navigation process.

Chapter 3
Learning of Environments

3.1 Ideas of Proposed Environment Learning Techniques

In the learning stage, we must construct an *environment map*, which includes information about the cameras, visiting targets, magnetic field, and obstacles. Then we can use such information in the processes conducted in the navigation stage, such as human localization and path planning. Specifically, we use a digital drawing of the floor plan of the environment to create the environment map, and specify the location of the cameras and the visiting targets in the map. A more detailed description of map construction will be introduced in Section 3.3.

After environment map construction, we continue to learn the magnetic field in the environment. The magnetic field is used for human orientation detection by the orientation sensor on the client device, and the output of the orientation sensor is an azimuth value specifying the orientation of the hand-held client device. In the magnetic field learning stage, we try to construct an *azimuth map*, which keeps a record of four-direction azimuth values for every sample location in the environment map.

Meanwhile, we also analyze the floor plan drawing of the environment to detect obstacles. The resulting obstacle information is used for collision avoidance in the path planning process. In the first step of obstacle analysis, we analyze the floor plan drawing to find the walkable regions in the environment, and detect accordingly the

obstacle regions. Next, we compute the orientations of the obstacles, which then are used in the path planning process to find proper moving directions at each spot for collision avoidance. These moving directions are called "*avoidance directions"* in the sequel of this thesis.

At last, we calibrate the cameras at both the server and client sides. For the server-side fisheye cameras, instead of calibrating the camera's intrinsic and extrinsic parameters, we adopt a space-mapping technique [12] for transformations between the coordinate systems used in this study, and extend the technique to be more flexible with better usability for our study. For the client-side camera on the mobile device, we introduce a simple technique to estimate the field-of-view of the camera, which then is used to map the locations of real-world objects onto the device screen. The proposed camera calibration scheme will be described in detail in Section 3.4.

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) Image coordinate system (ICS): denoted as (u, v). The u-v plane of this system coincides with the image plane of each fisheye camera and the origin is at the top-left of the image plane.

(2) Map coordinate system (MCS): denoted as (M_x, M_y). The MCS is used to represent the environment map. The M_x-M_y plane coincides with the image plane of the floor plan. The origin is at the left-top position of the image plane.

(3) Global coordinate system (GCS): denoted as (W_x, W_y, W_z). The W_x-W_y plane of

this system coincides with the ground and the *z* coordinates "grow to the top." The origin is at the left-top point in the MCS.

(4) Camera coordinate system (CCS): denoted as (*x*, *y*, *z*). The CCS is used to represent the real world space with respect to each fisheye camera. The *x* coordinates "grow to the right of the camera," the *y* coordinates "grow to the top of the camera," and the *z* coordinates "grow to the back of the camera." The origin is at the lens center of the camera.

In the proposed system, we use a floor plan drawing of the environment to establish the environment map, and the MCS is used for describing the geometry of the map, as mentioned previously. The relationship between the MCS and the GCS is illustrated in Figure 3.1. As shown in the figure, the origin of the MCS is mapped to a corresponding point in the real-world space. However, for the MCS the unit of pixel is used, so the global coordinates should be computed by multiplying the MCS coordinates by a scaling factor of the floor plan in the following way:

$$W_x = sM_x; \quad W_y = sM_y \tag{3.1}$$

where *s* is the scaling factor which is found by experiments.

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 stage. In Section 3.3.1, the information included in the environment map will be introduced briefly. And other information we need for the proposed system will be described in more detail in the subsequent

sections.

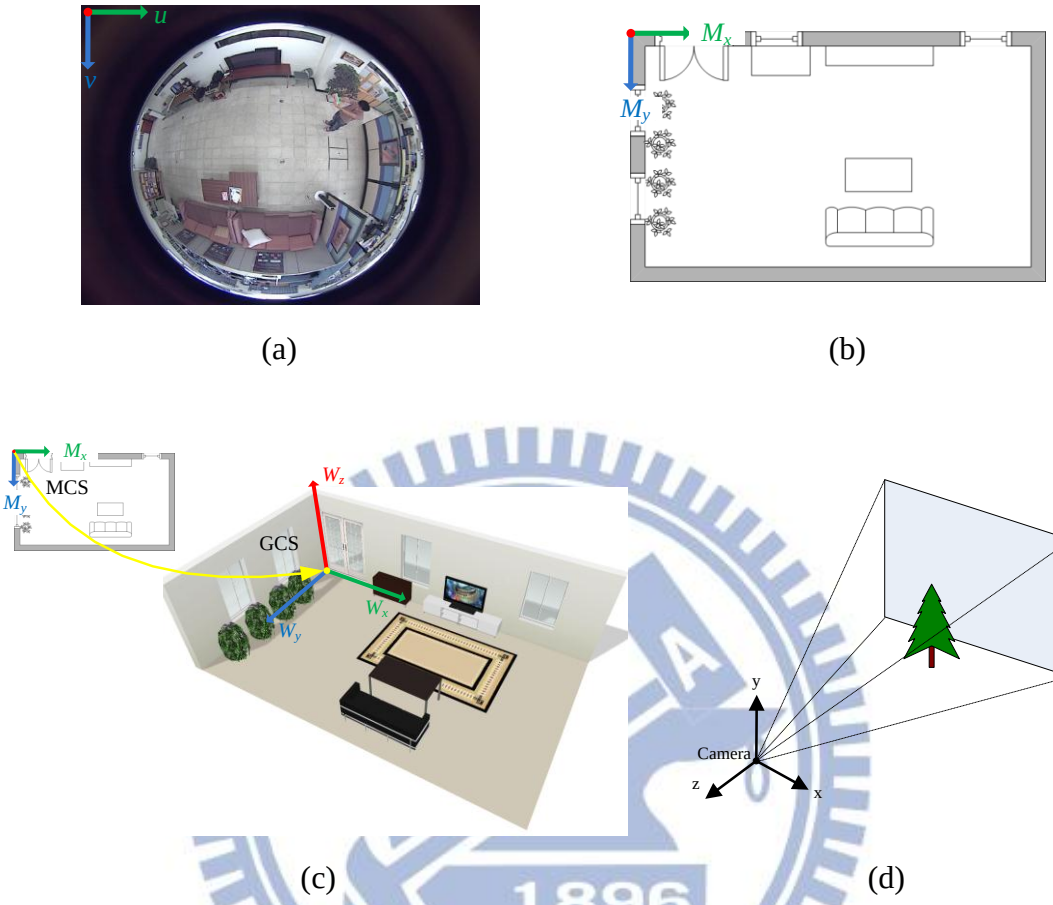


Figure 3.1 Four coordinate systems used in this study. (a) The ICS. (b) The MCS. (c) The relation between the MCS and the GCS. (d) The CCS.

3.3.1 Information of Environment Map

The information we include in the environment map includes the camera locations, visiting target information, obstacle information, and magnetic field information. The environment map we use is 2-D in dimension; in other words, it contains only one floor structure and 2-D coordinates. But the information contained in the map can be three-dimensional, that is, it includes height information.

The camera location specifies the position of a fisheye camera and its height. The

height of the camera is used for the transformation between the ICS and the GCS. In addition, a visiting target means a place or object in the real world of interest to visitors. We can place many visiting targets in the environment. Also, a user can search his/her desired destination by a keyword. The visiting target information includes the name of a visiting target, its coordinates in the GCS, and its *range*. The visiting target name is used for searching and displays. The location of the visiting target is specified by 3-D coordinates in the environment map, including its height. The range of the visiting target represents the visible region of the target in the real world. So the range is represented as a vertical plane in the real world, and a vertical plane is described by a height and a width.

The environment map includes the obstacle information as well. The obstacle information is used for path planning. It includes the regions and orientations of the obstacles in the environment. The obstacle region is used for collision detection, and the orientation is used for avoidance direction analysis. A more detailed description of path planning will be described in Chapter 5.

The last type of information included in environment map is the learned data about the magnetic field in the environment. The result of the magnetic field learning process is an azimuth map, which can be used for orientation detection. A more detailed description of the magnetic field learning process will be described in Section 3.3.4.

3.3.2 Finding Walkable Regions in Environment Floor Plan

The environment map is created from an image of the digital floor plan drawing, which we call the *floor plan image*. The floor plan image is a grayscale bitmap in

which walkable regions are drawn with white pixels. Also, we specify a scaling factor of the floor plan image as mentioned previously, which is used for computing the actual sizes of real objects from their sizes specified in unit of pixel in the image. Furthermore, if the floor plan drawing is a paper copy, then we can use a digital machine like a scanner or a camera to take a picture of it. For the experimental environment, the floor plan image we use is shown in Figure 3.2, which is created using the Microsoft Visio 2007 and exported as a bitmap.

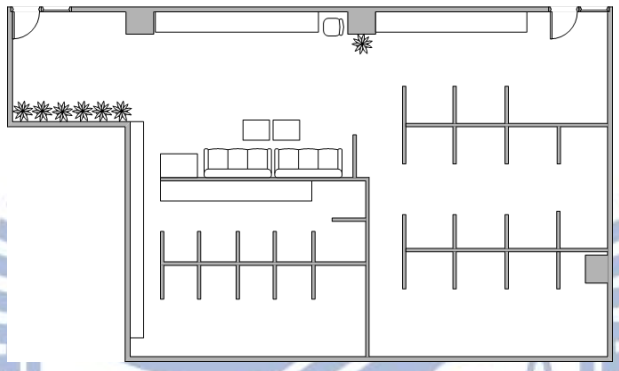


Figure 3.2 Floor plan image of the experimental environment map.

After we get a floor plan image, we can begin to construct the environment map. Here we assume that the walkable regions of the floor plan image are connected with white pixels, as mentioned previously. Then, we have just *one* walkable region which is the largest connected component in the image. Furthermore, we assume that the obstacle regions and the walkable region are separated by non-white pixels. Finally, we describe the algorithm we propose to find the walkable region from the floor plan image.

Algorithm 3.1 Finding the walkable region in the floor plan image.

Input: A floor plan image I_f, where the walkable region is the largest connected component and drawn with white pixels, and the obstacle regions and the walkable region are separated by non-white pixels.

Output: A binary walkable region image I_w, where the pixel values of the walkable region are specified by 1 and those of the obstacle regions by 0.

Steps

Step 1. Apply a threshold value t on I_f to get a temporary binary image I_{tmp}: if $I_f(x, y) > t$, then regard the pixel at (x, y) as white, and set $I_{tmp}(x, y)$ to 1; else, set $I_{tmp}(x, y)$ to 0.

Step 2. Find connected components in I_{tmp} using a connected component labeling algorithm.

Step 3. Select the maximum connected component C_{max} from the result of the last step as the walkable region.

Step 4. For all (x, y) in I_w, set $I_w(x, y)$ to 1 if C_{max} contains the pixel at (x, y); else, set $I_w(x, y)$ to 0.

3.3.3 Obstacle Orientation Analysis

In this section, we introduce the proposed obstacle orientation analysis scheme, which is used for obstacle avoidance in the path planning process. We can find avoidance directions from the walkable region image obtained by Algorithm 3.2. The details are as follows.

Algorithm 3.2 Finding avoidance directions.

Input: A walkable region image I_w, where the pixels of the walkable region are specified by 1 and those of the obstacle regions by 0.

Output: An obstacle avoidance map A.

Steps

Step 1. Get an obstacle image I_{obs} from the negative image of I_w

Step 2. Dilate the obstacle image I_{obs} to expand the obstacle regions

Step 3. Calculate the x and y derivatives of I_{obs} using the Sobel operator, resulting in two derivative maps D_x and D_y.

Step 4. Calculate the edge orientation and create an orientation map O by the following steps :

(1) set $O(x, y)$ to the angle between the vector $(D_x(x, y), D_y(x, y))$ and the vector (1, 0) if $D_y(x, y) \neq 0$ or $D_x(x, y) \neq 0$.

(2) set $O(x, y) = -1$, otherwise.

Step 5. Split O into small blocks, and for each block B_{ij} in O, construct a block orientation map O_b by the following steps:

(1) set $O_b(i, j) = m_{ij}$ if B_{ij} contains any non-negative value, where m_{ij} is the mean value of all non-negative values in B_{ij};

(2) set $O_b(i, j) = m'_{ij}$ if B_{ij} contains all negative values and the region of B_{ij} in O is all walkable, where m'_{ij} is the mean value of all non-negative values whose distances to the center of B_{ij} are smaller than a threshold d;

(3) set $O_b(i, j) = -1$, otherwise.

Step 6. Add $\pi/2$ to each element in O_b in the following way to get the obstacle avoidance map A:

$$A(i, j) = O_b(i, j) + \frac{\pi}{2}.$$

In order to make a planned path not too close to obstacles, we dilate the obstacle image to expand the obstacle regions in Step 2, and an example of the result is shown in Figure 3.3. If we collide with an obstacle, we may avoid it by going left or right. According to this concept, the avoidance directions are the vectors perpendicular to

the obstacle orientation vector.

In Step 5, we determine the values of O_b under three conditions based on the content of the obstacle image: 1) if a block B_{ij} contains edges of obstacle regions, then the resulting value is the mean of the angle values; 2) if B_{ij} is just the entire walkable region, the region of B_{ij} in O will be given all negative values, so the resulting value is set to the mean of the angle values around B_{ij}; and 3) if B_{ij} is just an entire obstacle region, it means we will never go to the region, so we do not have to compute the avoidance direction in such a region (marked by the value –1).

Finally, in Step 6, we add $\pi/2$ to get the angle of the avoidance directions. Therefore, each element in the map A represents one of two avoidance direction vectors. An example of the avoidance direction map yielded by the above algorithm is shown in Figure 3.4.

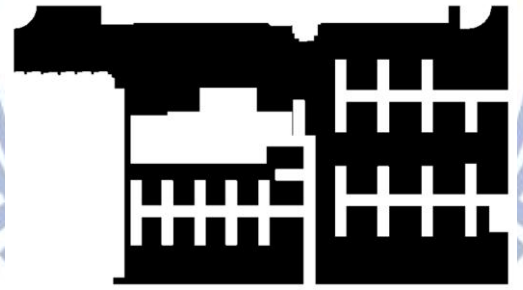


Figure 3.3 Expanded obstacle image of the experimental environment map where the white regions indicate the obstacle regions.

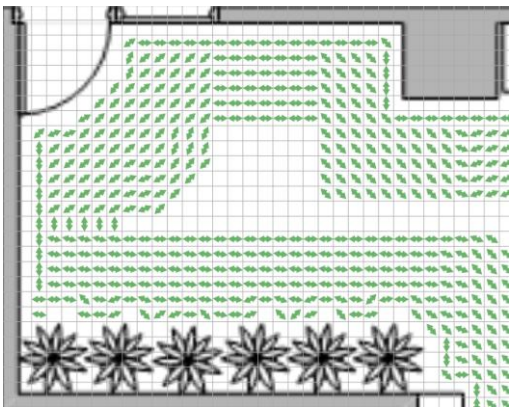


Figure 3.4 A part of the obstacle avoidance map of the experimental environment (shown in green arrows).

3.3.4 Learning of Magnetic Field Information

We can determine the human orientation by the orientation sensor on the client mobile device. The orientation sensor measures the azimuth angle of the device by detecting changes and disturbances in the magnetic field around the currently-visited spot. However, according to our experimental experience, the detected azimuth values are not stable enough for our application due to indoor magnetic interferences from various sources. The azimuth information is not used alone for human orientation detection in this study.

In the proposed magnetic field learning process, we measure the azimuth values at several sample points in the environment and construct an azimuth map, which then can be used for human orientation detection. The learning process is described as an algorithm in the following, where the four directions in the environment map are specified by direction vectors (1, 0), (0, 1), (–1, 0), and (0, –1).

Algorithm 3.3 Construction of an azimuth map for the experimental environment.

Input: Sample points *S* in the environment.

Output: An azimuth map *A*.

Steps

Step 1. Take the client device, and go to the first sample point S_0 at coordinates (x, y) in the environment map *A*.

Step 2. Face toward the direction (1, 0) in *A*, and measure the azimuth value a_0.

Step 3. Face toward the direction (0, 1) in *A*, and measure the azimuth value a_1.

Step 4. Face toward the direction (-1, 0) in *A*, and measure the azimuth value a_2.

Step 5. Face toward the direction (0, -1) in *A*, and measure the azimuth value a_3.

Step 6. Store the value set $(x, y, a_0, a_1, a_2, a_3)$ in *A*.

Step 7. Go to the next sample point and repeat Steps 2 through 6 until reaching the last sample point.

The above algorithm samples four azimuth values at every sample point. At each same point, four azimuths are measured for four different directions, respectively, each direction being perpendicular to the next one and the last perpendicular to the first one.

After we construct the azimuth map, we can use it to determine the human orientation. A more detailed description of such human orientation detection using the azimuth map will be described in Chapter 4.

3.3.5 Algorithm of Environment Construction

In this section, we summarize the processes described in the previous sections, as a total process — the process of environment construction, as described in Algorithm 3.4 below.

Algorithm 3.4 Construction of environment map for the experimental environment.

Input: A floor plan image *I*.

Output: An environment map *M*.

Steps

Step 1. Affix fisheye cameras onto the ceiling at proper locations in the environment.

Step 2. Create an environment map *M* by use of the floor plan image *I*, and specify a scaling factor.

Step 3. Specify the locations of the cameras on *M*.

Step 4. Specify the locations and the names of the selected visiting targets on *M*.

Step 5. Find the walkable region in *I* by Algorithm 3.1.

Step 6. Find obstacle regions and analyze avoidance directions in *I* by Algorithm 3.2.

Step 7. Construct an azimuth map by Algorithm 3.3.

3.4 Camera Calibration

In the section, we will describe the camera calibration processes proposed in this study. As mentioned previously, we use fisheye cameras to monitor the environment and analyze the omni-images to detect the human in the environment. In this study, we assume that there is only one human walking in the field of view of each fisheye camera. For the fisheye camera, we propose a space-mapping method for the transformation between the GCS and the ICS. Beside the fisheye cameras, the camera on the client mobile device must also be calibrated. A more detailed description of the camera calibration processes will be described in the following two sections.

3.4.1 Fisheye Camera Calibration and Ground Point Location Mapping

(A) Construction of a Calibration Box

Before the calibration process, we construct a calibration box first. The calibration box is a cube with four vertical planes and one horizontal plane. Each of the vertical and horizontal planes is called a *calibration board* in the subsequent sections. The calibration board is drawn to be of a chessboard pattern, which consists of 81 squares arranged in two alternating colors, namely, black and white. The central

square of the calibration board is drawn in the form of a cross pattern. Here we introduce a *calibration coordinate system* (CACS), which contains three coordinates (x, y, z) in the unit of cm. The origin of the CACS is located at the center of the central square of the horizontal calibration board. The x-y plane of the CACS is the aforementioned horizontal plane, and the z coordinates grow to the top. Therefore, we can obtain the coordinates of every corner point of the squares by the square size and the calibration board size. The calibration box and the CACS are shown in Figure 3.5.

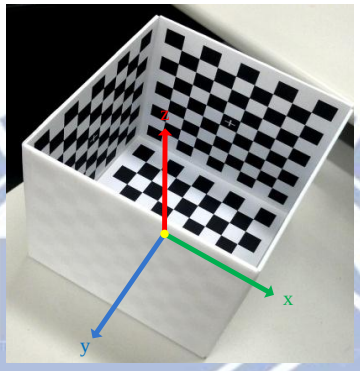


Figure 3.5 Calibration box and calibration coordinate system.

(B) Construction of Mapping Table

After the calibration box is constructed, we affix a fisheye camera to a ceiling spot right on the top of the calibration box, and make it look straight down at the cross of the central square of the bottom calibration board to capture a *calibration image*. An example of calibration images is shown in Figure 3.6(a). Then, we find the corners of all the squares in the calibration image (as shown in Figure 3.6(b)). Each corner is called a *calibration point* in the subsequent sections. Each calibration point is specified by their coordinates (u, v) in the ICS. As shown in Figure 3.7, each calibration point in the calibration image corresponds to one corner in the calibration box, so we can obtain a mapping table between the ICS and the CACS.

In order to perform the ground point location mapping, at first, the mapped calibration points of the vertical boards of the calibration box are projected onto the

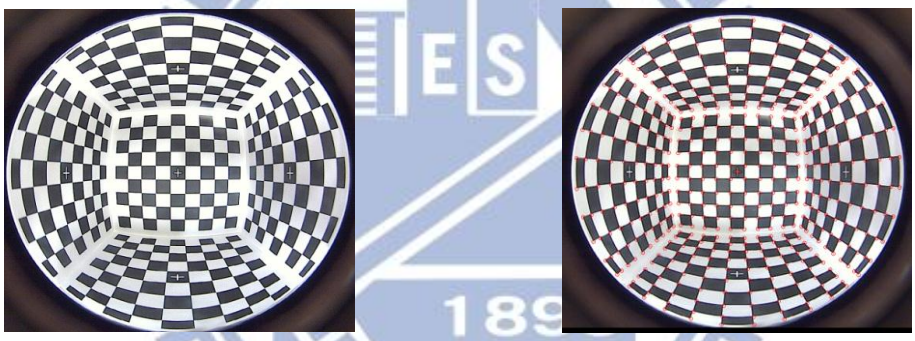
x-y plane of the CACS as shown in Figure 3.8. *C* is a calibration point on the calibration board with CACS coordinates (C_x, C_y, C_z), and C' is the projection point with CACS coordinates (C'_x, C'_y, 0). The relation between *C* and C' can be expressed by the following expression according to the principle of similar triangles:

$$\frac{H_c}{C_z} = \frac{C'_y}{C'_y - C_y} = \frac{C'_x}{C'_x - C_x}$$

Rearranging the above expression, we can get the coordinates of C' as:

$$C'_x = \frac{H_c \times C_x}{H_c - C_z}; \quad C'_y = \frac{H_c \times C_y}{H_c - C_z} \tag{3.2}$$

For the calibration points on the bottom calibration board, C' is identical to *C*.



(a) (b)

Figure 3.6 Calibration images. (a) The calibration captured from a fisheye camera. (b) The calibration points of the calibration image (shown as red circles).

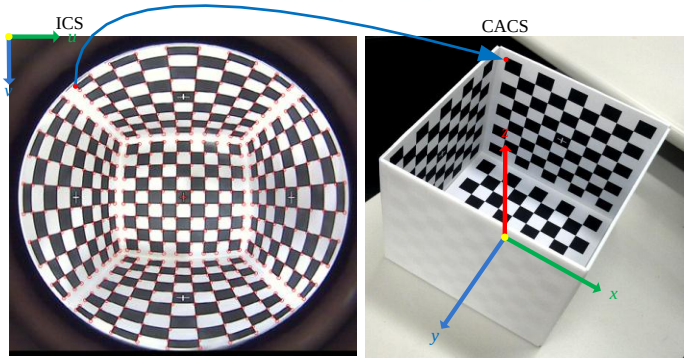


Figure 3.7 Mapping between the ICS and the CACS of a calibration point.

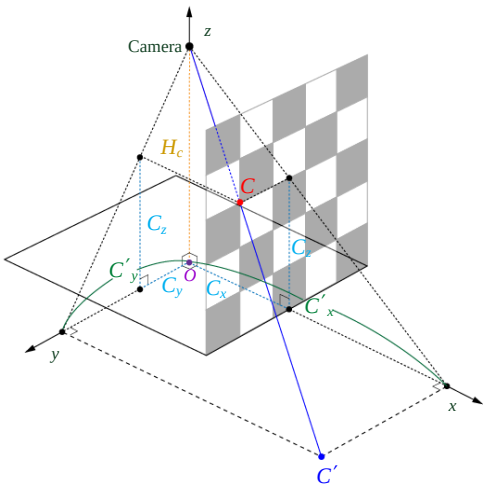


Figure 3.8 Projection of a point in CACS on the (x, y) plane, where H_c is the camera height, C is the calibration point on the calibration board, and C' is the projection point.

(C) Transformation using Mapping Table

So far, we have a mapping from the ICS to the CACS just for the calibration points only. For other points between them, we apply a bilinear interpolation scheme to get the corresponding mappings from the ICS to the CACS. As shown in Figure 3.9 Calculating the coordinates of a point between calibration points by bilinear interpolation., let p be a point in the ICS, and let A, B, C, and D be the four calibration points surrounding p. Denote s and t as the *relative distance* between the two end points within the range of 0 to 1. Suppose p is a point on the line $\overline{AB}$, the relative *distance* of p can be computed by $\left|\overline{Ap}\right| / \left|\overline{AB}\right|$. Then, we try to find the values of s and t, so that we can compute the CACS coordinates of p, as described in the following.

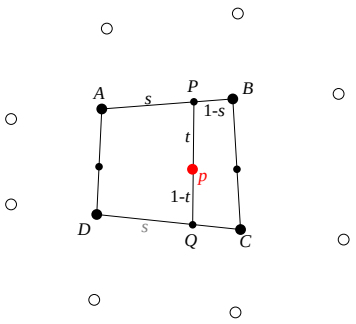


Figure 3.9 Calculating the coordinates of a point between calibration points by bilinear interpolation.

At first, *P* and *Q*, regarded as vectors of coordinates, can be expressed by the following expressions:

$$P = A + (B - A)s = A(1 - s) + Bs;$$

$$Q = D + (C - D)s = D(1 - s) + Cs.$$

And *p*, also regarded as coordinates, can be expressed by the following expression:

$$\begin{aligned} p &= P + (Q - P)t = P(1 - t) + Qt \\ &= A(1 - s)(1 - t) + Bs(1 - t) + Cst + D(1 - s)t \\ &= A(1 + st - s - t) + B(s - st) + Cst + D(t - st) \\ &= A + Ast - As - At + Bs - Bst + Cst + Dt - Dst \\ &= (1 - s)A + st(A - D + C - B) - t(A - D). \end{aligned} \tag{3.3}$$

So the two coordinates p_u and p_v can be expressed as:

$$\begin{aligned} p_u &= (1 - s)A_u + st(A_u - D_u + C_u - B_u) - t(A_u - D_u); \\ p_v &= (1 - s)A_v + st(A_v - D_v + C_v - B_v) - t(A_v - D_v). \end{aligned} \tag{3.4}$$

Rearranging the above expressions and substituting $U = A - D$, $V = A - D + C - B$, and $T = p - A$ into the rearranged results, we get

$$(T_u + tU_u) \times (tV_v + B_v - A_v) = (T_v + tU_v) \times (tV_u + B_u - A_u)$$

which may be reduced to be

$$t^2(U \otimes V) + t(T \otimes V + U \otimes (B - A)) + T \otimes (B - A) = 0$$

where ⊗ means the cross product operator. So the coefficients in the above quadratic equation may be derived to be:

$$\begin{aligned} a &= U \otimes V = (A - D) \otimes (A - D + C - B) = (A - D) \otimes (C - B); \\ b &= T \otimes V + U \otimes (B - A) = (p - A) \otimes (A - D + C - B) + (A - D) \otimes (B - A); \\ c &= T \otimes (B - A) = (p - A) \otimes (B - A). \end{aligned}$$

Finally, t may be solved to be:

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

where t will fall within the range of 0 to 1. Then, we may solve s by substituting t into Equation 3.5. After we obtain s and t, we can substitute the corresponding CACS coordinates for A, B, C, and D into Equation 3.3 to obtain the CACS coordinates of p finally.

At last, we want to transform the calibration points further from the CACS to the GCS to create a mapping from the ICS to the GCS for use in human localization (described later). As shown in Figure 3.10, we superimpose the calibration points on the images captured from the fisheye cameras, and then compute the location of pixels of the image by the mapping table. The heights of the cameras are specified in the environment map construction stage as mentioned in the previous section. As shown in Figure 3.11, a camera is affixed on the ceiling at a height of H, and G indicates the ground point of C'. By the principle of similar triangles, the distance d in the GCS can be computed by Equation 3.5. Accordingly, the coordinates (G_x, G_y) of G can be computed by the following equations:

$$d = \frac{H \times H_c}{d_c}; \tag{3.5}$$

$$\begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \frac{H \times H_c}{C'_x} \\ \frac{H \times H_c}{C'_y} \end{bmatrix} + \begin{bmatrix} p_x \\ p_y \end{bmatrix} \tag{3.6}$$

where p_x and p_y are the coordinates specifying the location of the camera in the GCS, and θ is the angle between the CACS axis and GCS axis (see Figure 3.12).

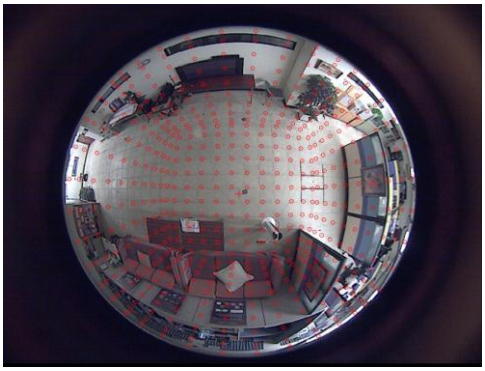


Figure 3.10 Superimposing calibration points on an omni-image.

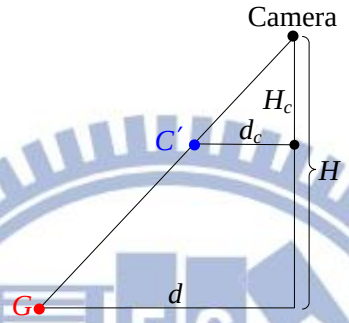


Figure 3.11 The projection of calibration point on the ground, where G is the projection point of C', and H is the height of the camera affixed on the ceiling.

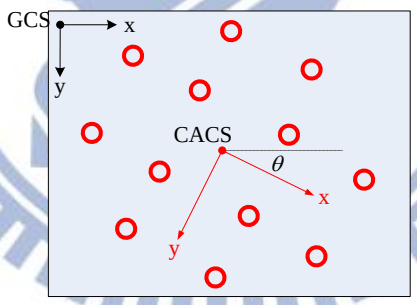


Figure 3.12 Angle between the GCS axis and the CACS axis. The red circles indicate the positions of calibration points.

3.4.2 Calibration of Camera on Mobile Device

Besides the fisheye camera, the camera on the mobile device must also be calibrated in order to get a projection matrix, by which points in the GCS may be projected onto the device screen.

In this study, a perspective camera model is used to represent the camera on the mobile device. In the theory of perspective projection [16], a 3D point in a view

frustum (as shown in Figure 3.13(a)) of the CCS is transformed into a unit cube (as shown in Figure 3.13(b)); the *x*-coordinate is transformed from the range [*l*, *r*] to the range [–1, 1], the *y*-coordinate is transformed from [*b*, *t*] to [–1, 1], and the *z*-coordinate is transformed from [*n*, *f*] to [–1, 1]. We can use a matrix *M* to perform the transformation from the *view frustum* to the unit cube in the following way:

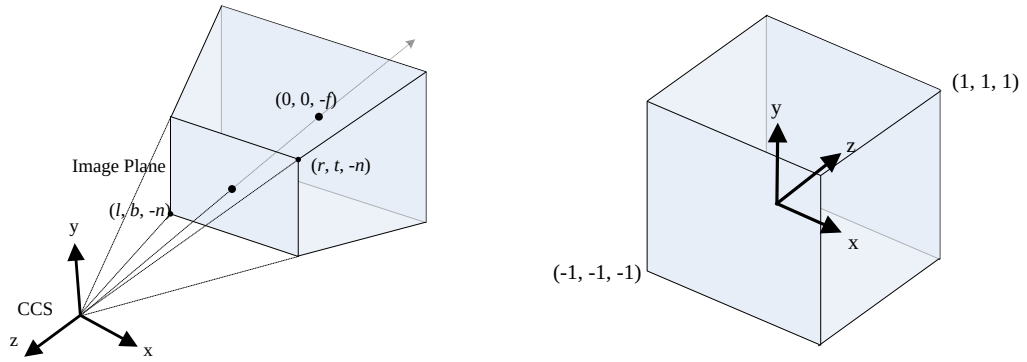
$$\begin{pmatrix} p'_x \\ p'_y \\ p'_z \\ p'_w \end{pmatrix} = M \cdot \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} \tag{3.7}$$

where p is a point in the view frustum, p' is the transformed point of p, and M is the projection matrix. In this transformation, the original coordinates of p' and p should be replaced by homogeneous coordinates, which have the fourth component *w*; and the nonhomogeneous coordinates *x*, *y*, *z* can be obtained by dividing by the *w*-component. The matrix M of Equation 3.7 can be expressed as follows according to [16]:

$$\begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

If the view frustum is symmetric, which means $r = -l$ and $t = -b$, then the matrix can be simplified to be:

$$\begin{pmatrix} \frac{n}{r} & 0 & 0 & 0 \\ 0 & \frac{n}{t} & 0 & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}. \tag{3.8}$$



(a) (b)

Figure 3.13 View frustum and unit cube. (a) The view frustum. (b) The unit cube.

In order to obtain this matrix, we need to determine the value of *r*, *t*, *n*, and *f*. As shown in Figure 3.14, the field-of-view of the camera in the *y* direction is denoted as α. Then, $\frac{n}{t}$ can be substituted by $\cot\frac{\alpha}{2}$. And the ratio $\frac{h}{w}$ of the image height *h* to the height *w* is known, so $\frac{n}{r}$ can be substituted by $\frac{h}{w}\cot\frac{\alpha}{2}$. Finally, the matrix of (3.8) can be expressed by:

$$\begin{pmatrix} \frac{h}{w}\cot\frac{\alpha}{2} & 0 & 0 & 0 \\ 0 & \cot\frac{\alpha}{2} & 0 & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}. \tag{3.9}$$

After obtaining the coordinates in the unit cube by Equation 3.7, we can obtain the coordinates (*u*, *v*) in the ICS by the following equations:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} w & 0 \\ 0 & h \end{pmatrix} \begin{pmatrix} 0.5 & 0 & 0 & 0.5 \\ 0 & -0.5 & 0 & 0.5 \end{pmatrix} \cdot \begin{pmatrix} p'_x / p'_w \\ p'_y / p'_w \\ p'_z / p'_w \\ 1 \end{pmatrix} \tag{3.10}$$

where *w* is the image width in the unit of pixel, and *h* is the image height. The values *n* and *f* of *M* only affect the resulting *z*-coordinate in the unit cube. According to the above equation, the z-coordinate does not affect the coordinates in the ICS. Actually,

the *z*-coordinate is used to represent the relationship of the distance of a point with respect to the viewpoint: the value of –1 means it is on the near plane of the view frustum, and 1 means it is on the far plane. Therefore, we can specify the value of both *n* and *f* arbitrarily, and then we have only one unknown variable α, which is the field-of-view in the *y* direction.

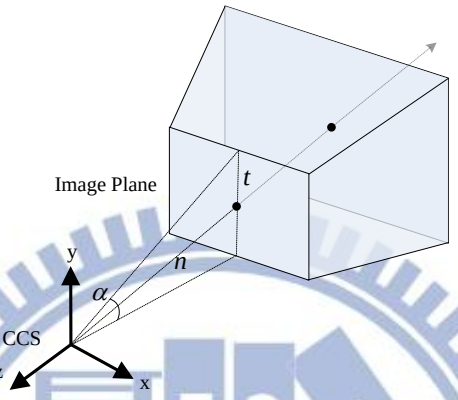


Figure 3.14 Field-of-view of the view frustum.

In this study, we also propose a simple method to estimate the value of α. As shown in Figure 3.15, we direct the camera to face toward the calibration board, which is drawn of the form of a grid pattern or others as long as it can help us measure the region. Then, we can observe the region which exactly fills the image (shown as the dark gray region). Accordingly, we can obtain the real width R_w and height R_h of the visible region by counting the grids or scale which are drawn on the calibration board. The distance d_R of the calibration board is known, so the value α, which defines the field of view in the *y* direction can be obtained by the following equation:

$$\alpha = 2\left(\tan^{-1}\left(\frac{R_h/2}{d_R}\right) \right). \tag{3.11}$$

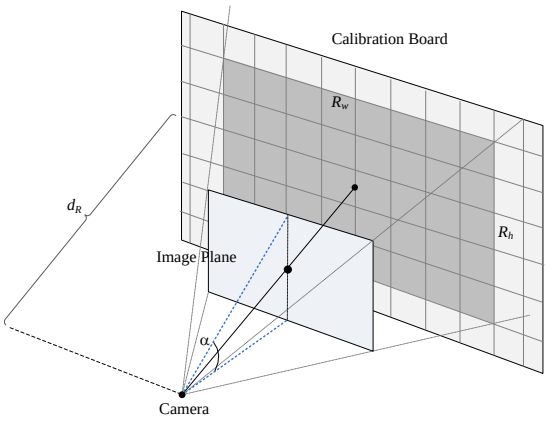


Figure 3.15 Finding the field of view angle α by measuring the visible region in image.

3.5 Experimental Results

An environment map of our experimental environment obtained by applying Algorithm 3.4 is shown in Figure 3.16. The scaling factor of the map is taken to be 40 pixels/m. The environment map includes eight visiting targets (shown as green regions) and two fisheye cameras (shown as blue circles). Images captured from the two cameras are shown in Figure 3.17.

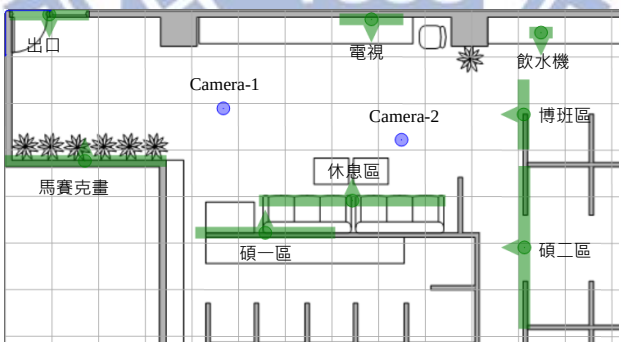


Figure 3.16 Environment map of the experimental environment, visiting targets are shown as green region, and cameras are shown as blue circles. The interval of the gray grid lines represents one meter in real world.

An obstacle avoidance map of the experimental map obtained by applying Algorithm 3.2 is shown in Figure 3.18, in which the avoidance directions are shown as green arrows. Blocks without avoidance directions mean that there are obstacle regions or regions which are away enough from obstacles.

(a) (b)

Figure 3.17 Images captured from the two fisheye cameras of the experimental environment. (a) An image captured from the Camera-1 of the map shown in Figure 3.16. (b) An image captured from the Camera-2.

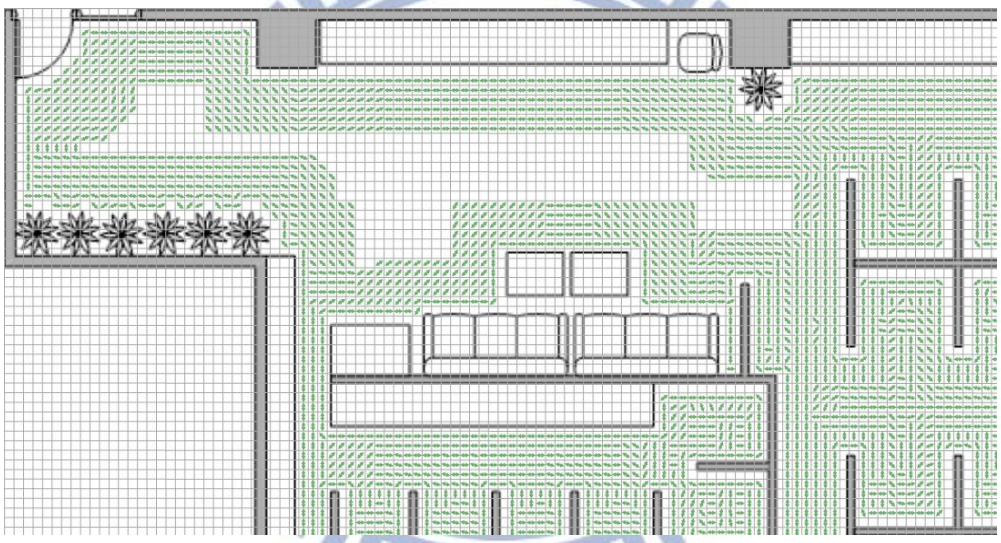


Figure 3.18 Obstacle avoidance map of the experimental environment.

Chapter 4
Human Localization in Indoor Environments by Computer Vision Techniques

4.1 Idea of Proposed Human Localization Techniques

In this study, we propose a human localization method using image-based analysis techniques for indoor environments. We have built a vision-based infrastructure with fisheye cameras affixed on the ceiling. The server-side system can access the omni-images captured with the cameras, and conduct detections of both the human location and orientation.

For human location detection, we perform background/foreground separation to detect the foreground image, and then apply connected component analysis to find the human activity region. Then, the user's foot point in this region is analyzed and transformed into the GCS. A more detailed description of the proposed human location detection scheme will be described in Section 4.2.

For human orientation detection, we use three different techniques integrally to obtain the orientation of the user. The first is the simplest way, which is to calculate the human motion by use of the human locations detected from consecutive video frames. The second is to use the orientation sensor on the client mobile device to detect the human orientation. The last is to attach a color edge mark on the mobile

device held by the human, and then analyze the omni-image to detect the color edge mark which is used to determine the orientation of the user. A more detailed description of the proposed human orientation detection scheme will be described in Section 4.3.

4.2 Human Location Detection

4.2.1 Background/Foreground Separation

The first step of the proposed human location detection scheme is background/foreground separation. As shown in Figure 4.1(a), we capture a background image before running the server-side system. When a user enters the environment, he/she will be considered as part of the foreground region (as shown in Figure 4.1 (b) and 4.1(c)). Therefore, we can obtain the human region by finding the connected components in the foreground image. Algorithm 4.1 below illustrates the steps to obtain the connected components in an omni-image.

Algorithm 4.1 Finding foreground regions in an omni-image.

Input: An omni-image I captured from a fisheye camera, a background image B captured beforehand, and a pre-selected threshold value T_D.

Output: Foreground regions in I.

Steps

Step 1. Subtract B from I to get a difference image D.

Step 2. Apply the threshold value T_D on D to get a foreground image F by the following steps:

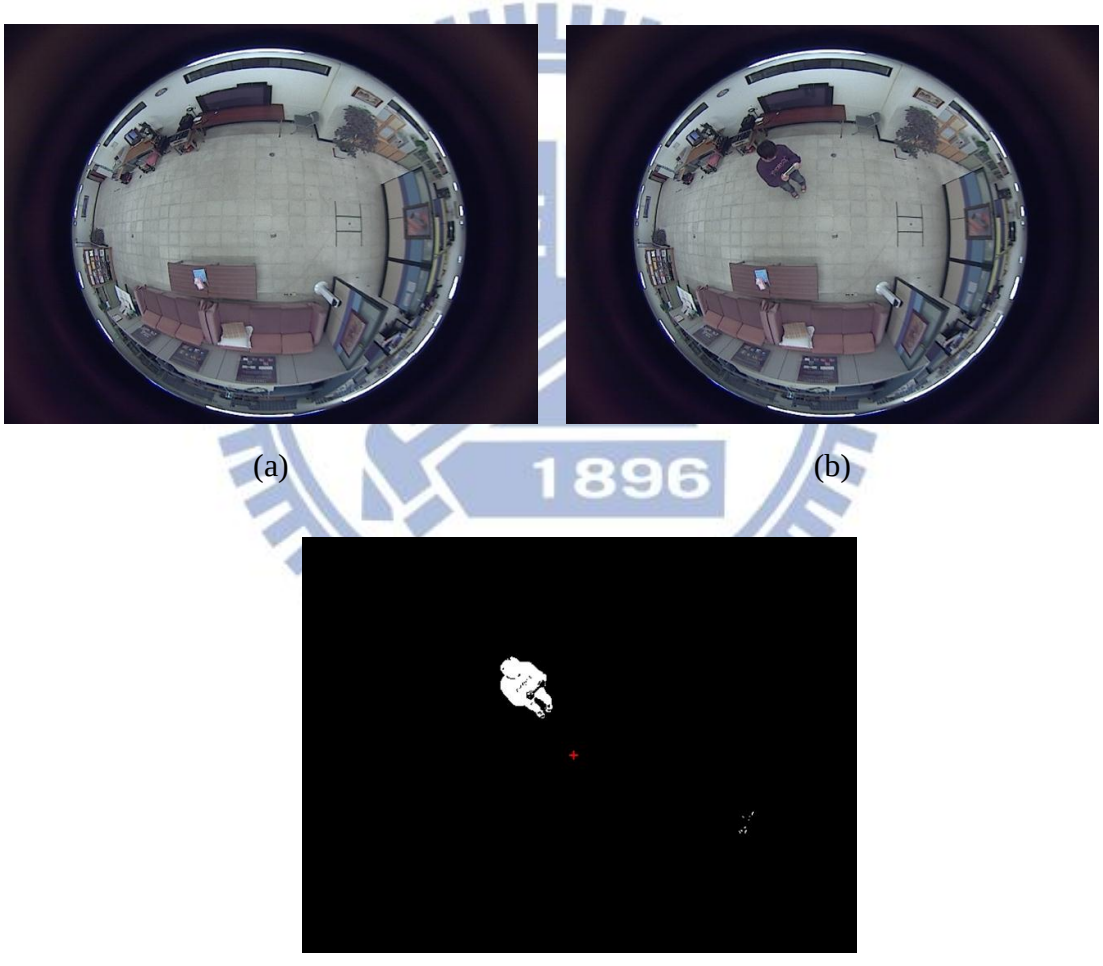
(1) set $F(u,v)=1$, if $|D(u,v)|>T_D$;

(2) set $F(u,v)=0$, otherwise.

Step 3. Apply the erosion operation to F to eliminate noise.

Step 4. Find connected components in F as the desired foreground regions using a connected component labeling algorithm.

In Step 3, we reduce noise by applying the erosion operation on the foreground image. However, the erosion operation will also eliminate the details of the foreground image. Another way to reduce noise is to set a larger threshold value in Step 2.



(a) (b)

(c)

Figure 4.1 Background/foreground separation. (a) The background image. (b) The image of the environment with a human. (c) The foreground image by subtracting (a) from (b).

4.2.2 Human Foot Point Detection and Computation

After obtaining the human region, we continue to find the foot point of the human in the region to determine the human location. For this aim, we use a property of the fisheye camera. With a fisheye camera affixed on the ceiling and looking straight down at the ground, a space line which is perpendicular to the ground will appear in the omni-image taken by the camera as a *radial line* passing through the image center, as shown in Figure 4.2. For example, the image lines of the edges of the pillar, door, bookcase, or wall in the environment will all appear to be so. We assume that the user using the proposed indoor AR navigation system is standing on the ground all the time, and so the axis of the user's body is perpendicular to the ground, meaning that the axis will go through the image center according to the above property.

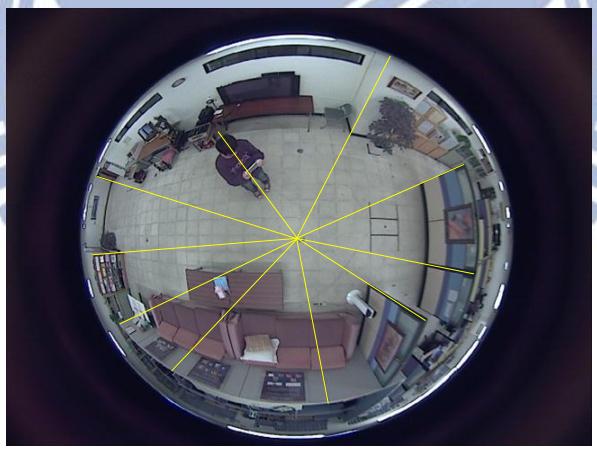


Figure 4.2 Extended image lines of space lines which are perpendicular to the ground will pass through the image center.

According to the above discussion, the foot point of a user can be obtained by finding the *nearest* point in the foreground region of the user *to* the image center. Here we assume that the foreground region of the user is the largest connected component R_{max} in the output of Algorithm 4.1. Therefore, we can find the user's location by the following algorithm using R_{max} as the input.

Algorithm 4.2 Computation of the human location.

Input: The foreground region R_{max} of a user.

Output: The user's location in the GCS.

Steps

Step 1. Find the nearest point f to the omni-image center in R_{max}.

Step 2. Project f onto the line $\overline{CC_R}$ to obtain a projection point f', where C is the omni-image center and C_R is the center of the bounding box circumscribing R_{max}.

Step 3. Transform f' into the GCS as output.

We want to let the foot point detected in Step 1 closer to the line going through the human's body axis. Therefore, according to the vertical line property mentioned previously, we project the detected foot point onto the human's body axis in Step 2. Finally, the human location can be computed by the spatial transformation described in Section 3.4.1. An example of the results is shown in Figure 4.3.

4.3 Human Orientation Detection

4.3.1 Orientation Detection by Human Motions

We detect a user's location on every omni-image, by which we can obtain a sequence of human locations, called the *location sequence*. Then, we use the sequence to compute the human moving orientation. However, the locations are detected by the previously-mentioned image-based technique, so the path composed of these locations may be not smooth. Therefore, if we just use the current human position and its previous one to calculate the motion vector, the resulting orientation will be unstable. In this study, we solve this problem by averaging all the motion vectors obtained

within a time interval.

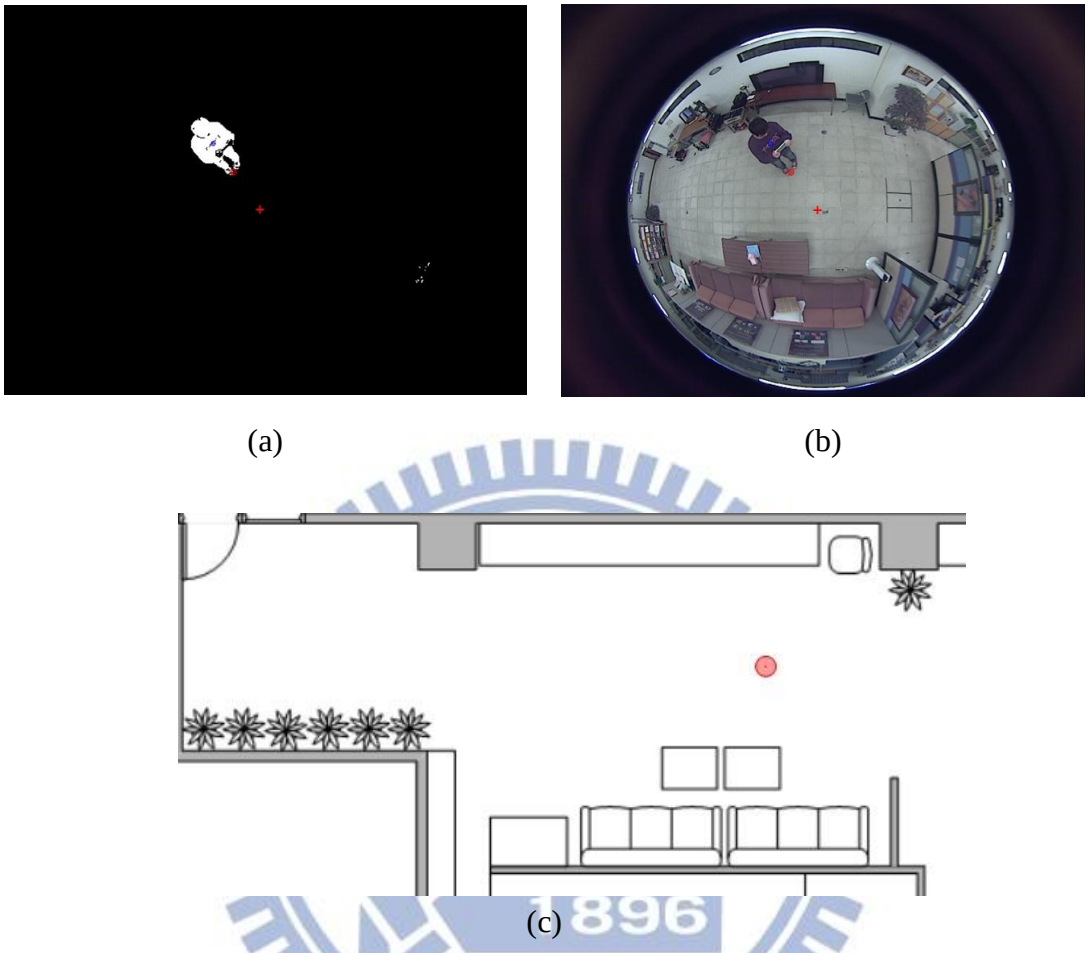


Figure 4.3 Detected foot point of a human (shown as red circle). (a) The foreground image. (b) The original image captured from the camera (c) The foot point in MCS.

However, the averaging operation will delay undesirably the orientation computation results when the human is turning. In other words, the human orientation will change quickly when the human is turning, but the averaging operation will cause the trend of the computed orientation changes to become slow. Therefore, we use a *turning flag* to determine whether a user is turning or not in the proposed human orientation detection scheme — if the current location of the user is on the right-hand side of the previous motion vector, we increment the turning flag by one; otherwise, we decrement it by one. But if the turning flag is negative when we increment the

turning flag, we will reset the turning to be zero. Symmetrically, we will reset it if it is positive when we decrement it. When the turning flag exceeds a threshold range, we will determine the human's turning direction by the sign of the turning flag, and then remove all the locations in the location sequence except the first one. In this way, the averaging result of motion vectors will be changing more quickly.

For example, as shown in Figure 4.4, the points p, P_1, and P_2 are all on the left-hand side of the previous motion vector. Assume that a turning flag f is zero at P_3. The turning flag f will be -3 at p. Then, we can determine that the human is turning to the left at p when T_f is smaller than 3. Consider another situation as shown in Figure 4.5. Assume that a turning flag f is zero at P_3. When the human reaches P_2, f will become -1, but when the human reaches P_1, f will be reset to be 0. So, the human will be determined to be walking forward instead of turning to the left at p.

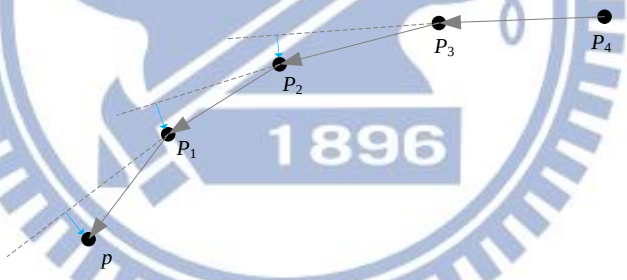


Figure 4.4 A path of turning to the left where each human foot point is on the left-hand side of the previous motion vector.

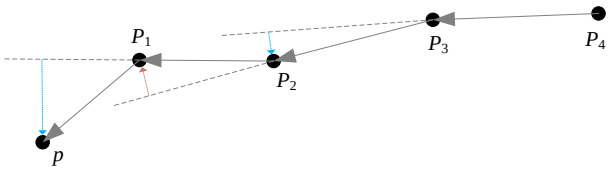


Figure 4.5 A path of walking forward where all points except P_1 are on the left-hand side of the previous motion vector.

The complete steps of human orientation detection by human motions are illustrated as an algorithm in the following.

Algorithm 4.3 Computation of the human orientation by human motions.

Input: The current location p of a user, a turning flag f, the location sequence P composed of the past human locations, where P_{i+1} means the previous location of P_i.

Output: The user's orientation $\overrightarrow{d_m}$.

Steps

Step 1. If the size of the sequence is larger than two, take the following steps.

1.1 Change the turning flag f by the following steps:

(1) increment f by 1, if $f \geq 0$ and p is on the right-hand side of the vector $\overrightarrow{P_2P_1}$;

(2) decrement f by 1, if $f \leq 0$ and p is on the left-hand side of the vector $\overrightarrow{P_2P_1}$;

(3) set f to be zero, otherwise.

1.2 If $f > T_f$ or $f < -T_f$, remove P_2, P_3, …, P_n from P, where T_f is a pre-selected threshold value.

Step 2. If the size of P is larger than a threshold size T_N, remove P_n from P.

Step 3. Compute the following direction vector $\overrightarrow{d_m}$ as output:

$$\overrightarrow{d_m} = \frac{1}{n}\left[\left(\sum_{k=1}^{n-1} P_k - P_{k+1}\right) + \left(p - P_1\right)\right].$$

Step 4. Insert p to the start of P.

In Step 1, we check the size of the location sequence to determine if there is a previous motion vector. If so, we can use it to detect the human's turning direction. In Step 2, in order to average the locations obtained in a certain time interval in the past,

we remove a redundant location in the location sequence if its size exceeds a threshold value. Finally, we compute the orientation by averaging the motion vectors and then insert the current location into the location sequence for the next cycle of orientation computation.

4.3.2 Orientation Detection by Orientation Sensor on Client Device

We determine the orientation of a user by his/her motions when his/her is moving. However, when the user is not walking, we seek another way to detect his/her orientation, which is through the use of the orientation sensor on the client mobile device. As mentioned in the previous sections, the orientation sensor measures the azimuth angle of the device by detecting changes and disturbances in a magnetic field. In Chapter 3, we have established an azimuth map for orientation detection. The following algorithm describes how we determine the orientation using the azimuth map.

Algorithm 4.4 Computation of the human orientation by the orientation sensor.

Input: The current location p of a user, the azimuth value a detected by the client device held by the user, and an azimuth map A.

Output: The user's orientation $\overrightarrow{d_o}$.

Steps

Step 1. Find the nearest sample point A_p to p in A, and get a value set of A_p, (x, y, a_0, a_1, a_2, a_3), which contains the azimuth values of the four major directions as described in Section 3.3.4.

Step 2. Find the two azimuth angles a_n and $a_{(n+1) \bmod 4}$ of A_p which a is in between, by finding a parameter n satisfying the following two constraints:

(1) $V(a)$ is on the right-hand side of $V(a_n)$ where the function $V(\theta)$ returns a vector with the angle θ, i.e., $V(\theta) = \overrightarrow{(\cos\theta,\ \sin\theta)}$;

(2) $V(a)$ is on the left-hand side of $V(a_{(n+1) \bmod 4})$.

(The relation between $V(a)$, $V(a_n)$, and $V(a_{(n+1) \bmod 4})$ is shown in Figure 4.6.)

Step 3. Compute the relative position r of a between a_n and $a_{(n+1) \bmod 4}$ by the following equations:

$$r = \frac{|P(a) - P(a_n)|}{|P(a_{(n+1) \bmod 4}) - P(a_n)|}.$$

where the function $P(\theta)$ returns a point with the coordinates $(\cos\theta, \sin\theta)$.

Step 4. Compute the direction vector $\overrightarrow{d_o}$ as output by interpolation:

$$\overrightarrow{d_o} = r \cdot \overrightarrow{v_{(n+1) \bmod 4}} - (1-r) \cdot \overrightarrow{v_n}$$

where $\overrightarrow{v_x}$ is the corresponding direction vector with respect to the azimuth a_x, which is defined previously in the learning stage.

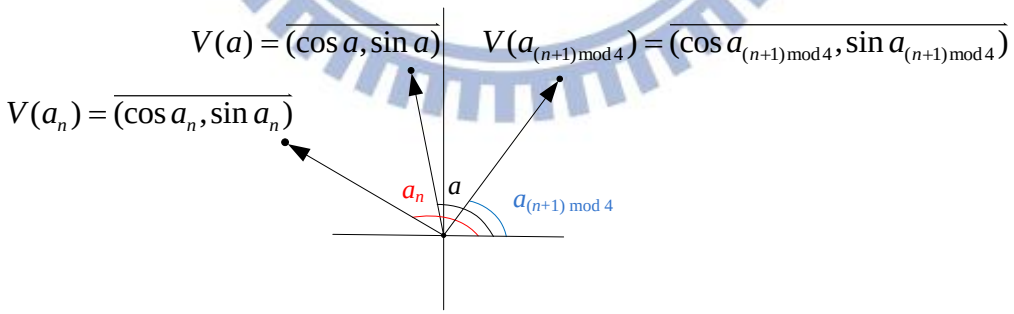


Figure 4.6 A azimuth a between two azimuth a_n and $a_{(n+1) \bmod 4}$, where $V(a)$ is on the right-hand side of $V(a_n)$ and on the left-hand side of $V(a_{(n+1) \bmod 4})$.

Because the sample points in the azimuth map are discrete, we choose the nearest one in Step 1 to approximate the result. In Step 2, we find the region which the

azimuth value *a* falls on. The region is between two azimuth values at the sample point we choose, so we obtain the final orientation vector by interpolation as done in Steps 3 and 4.

4.3.3 Orientation Detection by Color Edge Mark on Top of Client Device

We introduced two orientation detection techniques in the previous sections. However, they still have the stability and precision problems, which cause failures in detecting the human orientation. Specifically, detection by human motions can only be used when the human moving, and detection by orientation motions is not stable enough due to magnetic interferences almost everywhere.

Figure 4.7 The color edge mark (The green strip) in the omni-image.

Here we propose a technique to solve the problems by attaching a *color edge mark* on the top of the client device. The color of the color edge mark has high saturation and high lightness, so it can be segmented easily from the omni-images. As shown in Figure 4.7, we can see the green edge color edge mark clearly in the omni-image.

In order to separate the color edge mark from an omni-image, at first we convert the color space of the omni-image from the RGB color space to the HSV one. The HSV

color model assigns three color components to a pixel, which respectively are *hue*, *saturation*, and *value*. The *hue* is described with the words we normally think of as describing color: red, blue, green, etc. The *saturation* refers to the dominance of hue in the color. The *value* is the lightness of the color. By the HSV color model, we can separate the color edge mark from the omni-image more easily, because the color of the color edge mark has high saturation and high lightness.

The color edge mark becomes a strip shape in omni-images, so we can apply a line approximation scheme to the detected color edge mark. Then, we try to compute the direction vector of the approximating line in order to determine the device orientation. Under the assumption that the user holds the device horizontally, the color edge mark becomes parallel to the ground. As shown in Figure 4.8, the color edge mark is represented as a solid green line, and the red line and the solid green line are projected onto identical image points; meanwhile, the vertical projection (shown as the dotted green line) of the color edge mark is parallel to the red line. Therefore, we can determine the orientation of the color edge mark by the orientation of the red line.

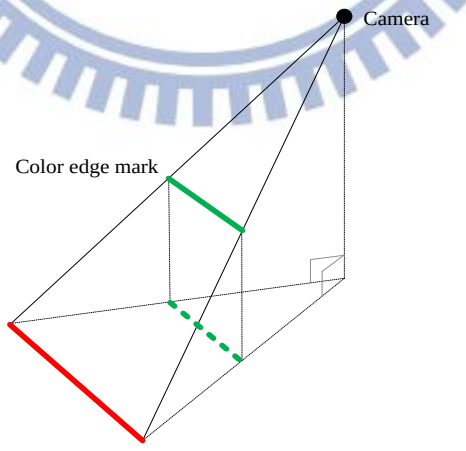


Figure 4.8 The red line and the color edge mark (shown as solid green line) are projected onto identical image points. The vertical projection (shown as dotted green line) of the color edge mark will be parallel to the red line.

The following algorithm describes the process to detect the human orientation by the color edge mark.

Algorithm 4.5 Computation of the human orientation by the color edge mark on top of the client device.

Input: An omni-image *I*, the foreground region *R* of a user, and the orientation vector $\overrightarrow{d_o}$ detected by Algorithm 4.4.

Output: The user's orientation, $\overrightarrow{d_c}$, and a reliability index r_c.

Steps

Step 1. Create an image I_{hsv} by converting the color space of *I* from RGB to HSV by the following equations:

$$V = \max(R, G, B);$$

$$S = \begin{cases} \dfrac{V - \min(R, G, B)}{V} & \text{if } V \neq 0; \\ 0 & \text{otherwise;} \end{cases}$$

$$H = \begin{cases} \dfrac{(G - B) \times 60}{S} & \text{if } V = R; \\ 180 + \dfrac{(B - R) \times 60}{S} & \text{if } V = G; \\ 240 + \dfrac{(R - G) \times 60}{S} & \text{if } V = B. \end{cases}$$

If $H < 0$, then increment H by 360^o.

Step 2. Create a binary image I_c, and for all (u, v) in I_c, assign values to $I_c(u,v)$ by the following steps:

(1) set $I_c(u,v) = 1$, if the value of $I_{hsv}(u,v)$, (h, s, v), is between two threshold values $(H_{min}, S_{min}, V_{min})$ and $(H_{max}, S_{max}, V_{max})$;

(2) set $I_c(u,v) = 0$, otherwise.

Step 3. Find the connected components *C* in region *R* of I_c: if there has no

connected component in region *R*, set the reliability index r_c to zero and end the algorithm.

Step 4. Find the bounding box *B* of *C*.

Step 5. Apply a line approximation algorithm to the pixels in *C*, resulting in an approximated line *L*.

Step 6. Find the two intersection points *p* and *q* of *B* and *L*.

Step 7. Get *s* and *t* by converting *p* and *q* from the ICS to the GCS.

Step 8. Compute the orientation $\overrightarrow{d_c}$ as output by the following steps:

(1) set $\overrightarrow{d_c} = (-\overrightarrow{st}_y, \overrightarrow{st}_x)$, if $(-\overrightarrow{st}_y, \overrightarrow{st}_x) \cdot \overrightarrow{d_o} < (\overrightarrow{st}_y, -\overrightarrow{st}_x) \cdot \overrightarrow{d_o}$;

(2) set $\overrightarrow{d_c} = (\overrightarrow{st}_y, -\overrightarrow{st}_x)$, otherwise.

Step 9. Set the reliability index r_c to be the size of *C*.

We apply two threshold values to extract the region of color edge mark in Step 2, and an example of the result is shown in Figure 4.9(a). We will get two orientation vectors which are perpendicular to the vector of the color edge mark, and we have to determine which one is correct. Here we use the orientation vector $\overrightarrow{d_o}$ detected by Algorithm 4.4 to make a decision by choosing the one which is closer to $\overrightarrow{d_o}$.

Furthermore, the color edge mark may not be seen in an omni-image due to obstacle covering or a long distance away from the camera. We set a reliability index to be the size of the region of the color edge mark for making a decision about whether the color edge mark will be used or not. A larger value indicates a better visibility of the color edge mark.

In addition, as mentioned in the previous section, we choose the foreground region with the largest region as the human region. Here we can also determine this

region by checking whether a color edge mark is detected or not.

Figure 4.9 shows an example of the results of applying Algorithm 4.5.

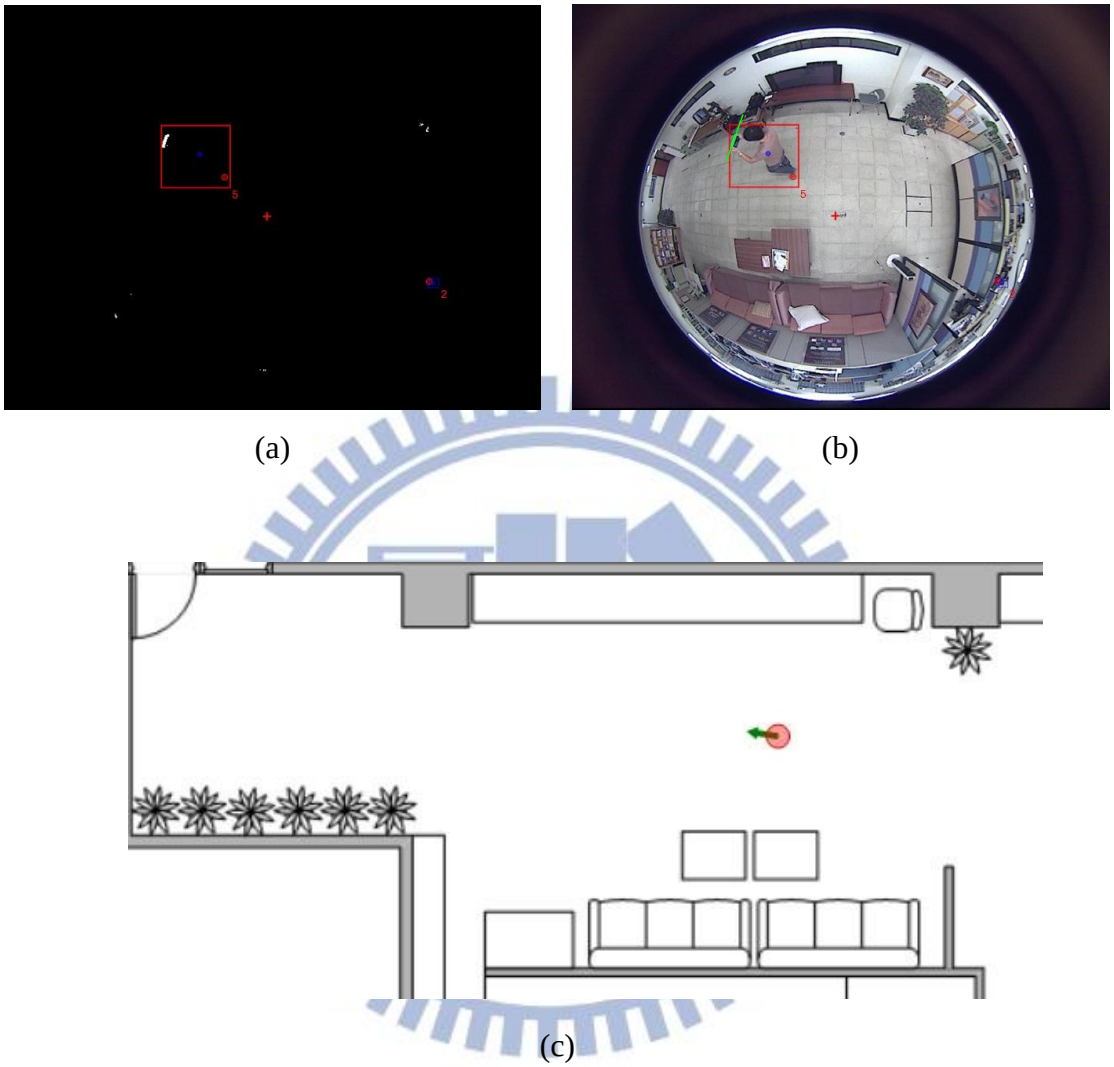


Figure 4.9 Orientation detection by color edge mark on top of the mobile device. (a) The color edge mark region segmented from the omni-image. (b) The approximating line (shown as green) obtained by applying line approximation on (a). (c) The detected orientation (shown as green).

4.3.4 Algorithm of Orientation Detection

We introduced three different techniques for orientation detection in the previous sections. In this section, we will describe how we integrate the three techniques to

perform the orientation detection work more reliably.

Algorithm 4.6 Computation of the human orientation.

Input: A user's location, the foreground region h_u of the user, and the omni-image which contains h_u.

Output: The user's orientation $\vec{d}$.

Steps

Step 1. Compute the orientation $\overrightarrow{d_o}$ by Algorithm 4.4.

Step 2. Use $\overrightarrow{d_o}$ to compute the orientation $\overrightarrow{d_c}$ by Algorithm 4.5, resulting a reliability index r_c.

Step 3. If $r_c > T_r$, where T_r is a threshold value, then set $\vec{d}$ to be $\overrightarrow{d_o}$ as the output and finish this algorithm.

Step 4. If the human is walking, compute the orientation $\overrightarrow{d_m}$ by Algorithm 4.3, and set $\vec{d}$ to be $\overrightarrow{d_m}$ as the output. Otherwise, set $\vec{d}$ to be $\overrightarrow{d_o}$ as the output.

4.4 Human Tracking

4.4.1 Idea of Human Tracking

The objective of human tracking is to identify the same human in consecutive video frames. Consecutive video frames come from the omni-images captured from the fisheye cameras in the proposed system. In this study, we adopt a human tracking method, which is called *high level tracking*, proposed by Newman, et al. [17]. In

Algorithm 4.1, we find the foreground regions in a foreground image. Then, we find a bounding box for each foreground region. For each successive frame, the human tracking algorithm *associates* a foreground region with one of the existing *tracks*. A *track* represents an identical object moving in consecutive video frames. This is achieved by constructing a *tracking matrix* representing the distance between each of the foreground regions and all the existing tracks. Each row of the tracking matrix corresponds to one track, and each column corresponds to one foreground region. The distance is computed using a *bounding box distance measure* proposed by the adopted method by Newman, et al. [17]. As shown in Figure 4.10(a), the distance between bounding boxes *A* and *B* is the lower of 1) the distance from the center of *A* to the nearest point on *B* and 2) that from the center of *B* to the nearest point on *A*. If either center lies within the other bounding box (as shown in Figure 4.10(b)), then the distance is zero.

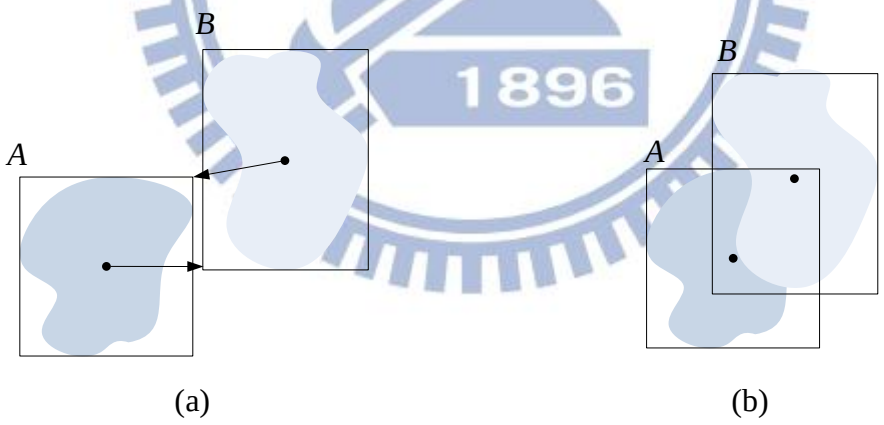
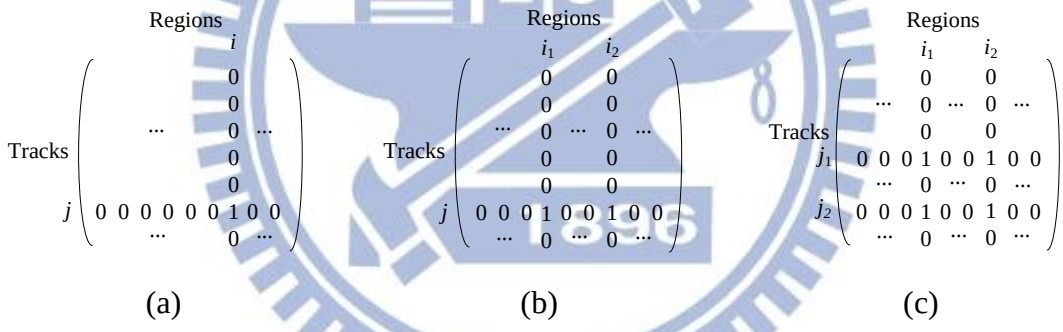


Figure 4.10 The bounding box distance measure. (a) The distance between *A* and *B* is the lower of the distance from the center of *A* to the nearest point on *B* or from the center of *B* to the nearest point on *A*. (b) The distance is zero.

If we consider a region is close enough to a track, then the value, which is at the corresponding column of the region and the row of the track, is incremented by one. If a foreground region is close enough to only one track and only a region is close

enough to the track, i.e., if it is a one-to-one correspondence, then the column and the row will both have one “one” (as shown in Figure 4.11(a)), so we may associate the region with the track. However, two regions may be both close enough to one track, and this will cause two “one” at a row (as shown in Figure 4.11(b)), then we associate both the two regions to the track. Similarly, if a region is close enough to two tracks, we associate the region with both of the two tracks. If two regions are close enough to two identical tracks (as shown in Figure 4.11(c)), then we associate the two regions with both of the two tracks. Based on the above concepts, we can associate regions with existing tracks, by which we can identify the same human in consecutive video frames.



(a) (b) (c)

Figure 4.11 Tracking matrix at different situation. (a) A region is close enough to only a track, and only one region is close enough to the track. (b) Two regions are close enough to a track. (c) Two regions are close enough to two same tracks.

The human tracking algorithm using the adopted method is described in Algorithm 4.7.

Algorithm 4.7 Human tracking.

Input: The foreground regions *C* in a frame and a set of tracks *T*, with each track being associated with at least one foreground region, T_i meaning the *i*th track

in T, and C_i meaning the ith foreground region in C.

Output: Tracks T

Steps

Step 1. Create a tracking matrix M with all zero's with each row of M corresponding to one track of T, and each column of M corresponding to one region of C.

Step 2. Compute the bounding box distance d_{ij} between each of C_i and $R(T_j)$ using the bounding box distance measure where the function $R(t)$ returns the associated region of the track t.

Step 3. For each d_{ij}, set $M(i, j)$ to be 1 if $d_{ij} < T_D$, where T_D is a pre-selected threshold value.

Step 4. Perform the following steps for M.

4.1. For each column i with only one non-zero element at row j, and row j has only one non-zero element at column i, associate C_i with T_j.

4.2. For each column i with all zero elements, create a new track t_{new}, associate it with C_i, and add t_{new} into T.

4.3. For each row j with all zero elements, remove T_j from T.

4.4. For the columns $i_1, i_2, \ldots, i_m$ which have more than one non-zero elements at rows $j_1, j_2, \ldots, j_n$, associate $C_1, C_2, \ldots, C_m$ with $T_1, T_2, \ldots, T_n$.

In Step 3, we binarize the tracking map by the resulting distance; if two bounding boxes are close enough, the resulting value is one; otherwise, it is zero. In Step 4.2, if a foreground region is not associated with any track, then it is regarded as a new object which we have to track. We remove tracks which are not associated with any object in Step 4.3.

4.4.2 Camera Hand-off

The fisheye camera has a wider field-of-view, and can observe a wider range than normal cameras. However, if an object is located outside the view of a fisheye camera, or far away from a fisheye camera, it will be projected onto a small region in the omni-image captured from the camera. Therefore, the number of pixels of the small region will be too small to perform analysis mentioned in the previous sections. In order to track the user around the entire environment, we need more than one camera to monitor the entire environment. When we have multiple cameras, a user may "appear" in more than one omni-image captured from different cameras. Therefore, we have to determine which camera we should use and continue to track the same user between different cameras, and this is the so-called *camera hand-off* problem.

To handle the problem, in each human localization and tracking processing, we obtain a foreground region representing a user. The foreground region belongs to an omni-image captured from a certain camera. If we have a foreground region obtained in the previous processing work, then we can obtain the foreground region representing the same user in the current processing by the following algorithm.

Algorithm 4.8 Camera hand-off.

Input: The foreground region h_u of a user, and the tracks $T(,)$, where $T(i,)$ means all tracks in omni-image I_i captured from the camera C_i, and $T(i,j)$ means the jth track in $T(i,)$.

Output: The human region of the user in the current process work.

Steps

Step 1. Find the track $T(s,u)$, which is associated with the foreground region h_u

Step 2. Set h_s to be the associated region of $T(s,u)$.

Step 3. Compute the location p_s of h_s in the GCS by Algorithm 4.2.

Step 4. Set $T' = \bigcup_{i \neq s} T(i,)$.

Step 5. Compute the location in the GCS for all foreground regions associated with T', resulting in a location set *P*.

Step 6. Find the foreground region p_t with its location p_t which is associated with the track $\in T(t,)$ and satisfies the following two constraints:

(1) $d(p) = \left|\overline{pp_s}\right|$ $p \in P$ is the minimum for $p = p_t$.

(2) $\left|\overline{p_t p_s}\right| \leq \lambda$, where λ is a pre-selected threshold value.

Step 7. If p_t is found, compute the distance $d_t = \left|\overline{p_t c_t}\right|$ and $d_s = \left|\overline{p_s c_s}\right|$, where c_t is the location of the camera C_t and c_s is the location of the camera C_s. If p_t is not found, set h_s as output and finish this algorithm.

Step 8. Set h_s as the output if $d_s < d_t$; otherwise, set h_t as the output.

The algorithm finds a foreground region representing the same object as the input foreground region. If we find a foreground region representing a user in the previous processing work, then we can use the region as input to find the corresponding region in the next process work.

At first, we find the track which is associated with the input region. The found track is called a "*user track*". And then we find a region which is closest to the region associated with the user track in Step 4 through 6. Then, we compare each of the locations of the two regions with each of the locations of its corresponding cameras. Finally, the one closer to its corresponding camera is chosen to be the output.

4.5 Algorithm of Human Localization and Tracking

In this section, we will describe the complete steps for human localization and tracking. Under the assumption that there is only one user in the indoor environment taken care of by the system, we can obtain one foreground region representing a user for each processing cycle. Therefore, the server-side system will send the location and orientation to the user's client-side system. The following algorithm illustrates the complete steps for this task.

Algorithm 4.9 Algorithm of human localization and tracking.

Input: The foreground region h_u of a user in the previous processing cycle, the omni-images I_1, I_2, …, I_n captured from cameras C_1, C_2, …, C_n, respectively, where n is the number of the cameras.

Output: The user's location p, the user's orientation $\vec{d}$, and the foreground region h'_u of the user.

Steps

Step 1. Find foreground regions $R(,)$ in each of I_1, I_2, …, I_n by Algorithm 4.1, where $R(i,)$ means all foreground regions in omni-image I_i, and $R(i, j)$ means the jth foreground region in $R(i,)$.

Step 2. Track the foreground regions $R(1,)$, $R(2,)$, …, $R(n,)$ by Algorithm 4.7, resulting in the tracks $T(1,)$, $T(2,)$, …, $T(n,)$, where $T(i,)$ means all tracks in omni-image I_i, and $T(i, j)$ means the jth track in $T(i,)$.

Step 3. If h_u is set, then find h'_u by Algorithm 4.8; else, take the following steps.

3.1. Detect the color edge mark by Algorithm 4.5 for $R(1,)$, $R(2,)$, …, $R(n,)$.

3.2. If the color edge mark is detected in any foreground region, apply Algorithm 4.8 to anyone of the foreground regions with color edge mark detected to find the human region h'_u and go to Step 4.

3.3. If the color edge mark is not detected, apply Algorithm 4.8 to the foreground region with the largest region size in *R* to find the human region h'_u and go to Step 4.

Step 4. Compute the location *p* of h'_u as the output by Algorithm 4.2.

Step 5. Compute the orientation $\vec{d}$ of h'_u as the output by Algorithm 4.6.

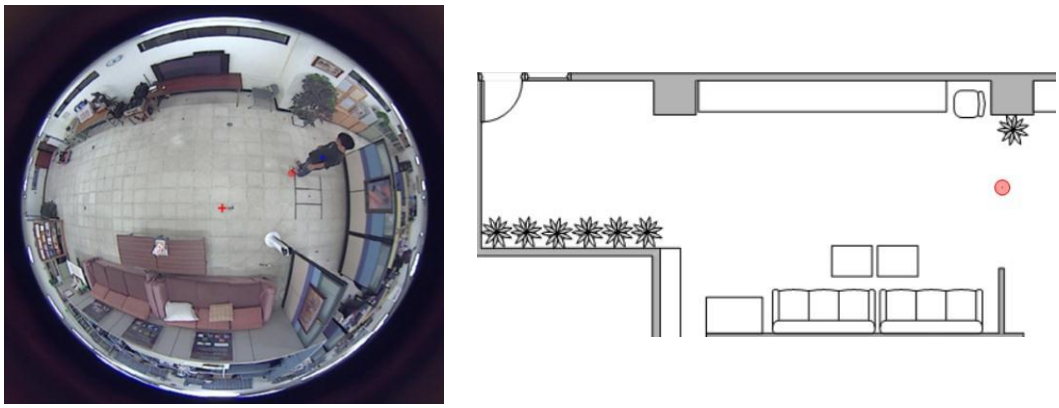
In Step 3, if we have found a human region in the previous processing cycle, we can find the same human region by performing the camera-off algorithm. Otherwise, we find the human region by the color edge mark or the region size. Finally, we compute the location and the orientation of the human region as output. Furthermore, the human region will be used in the next processing cycle.

4.6 Experimental Results

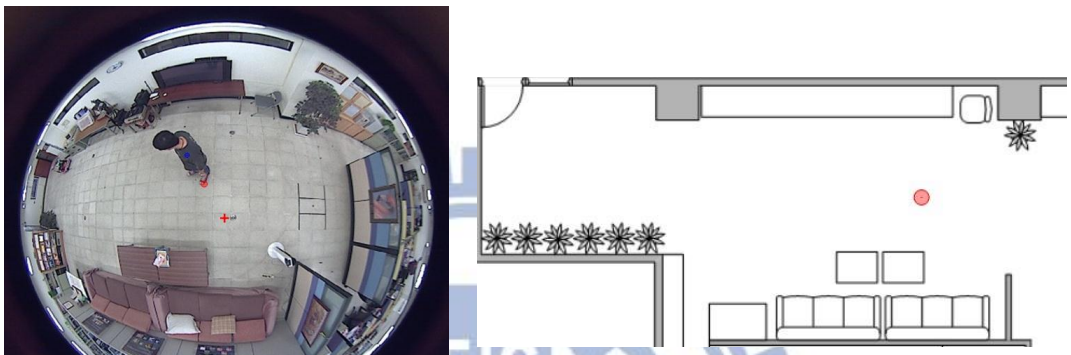
In this section, we show some experimental results of both human location detection and orientation detection. Figure 4.12 shows the results of the human location detection at four different locations. It shows that the proposed method can actually find the foot point of a human and transform the point from the ICS to the MCS.

Figure 4.13 shows the results of the human orientation detection by the color edge mark, where the approximating lines are shown as light green color. It also shows that the proposed method is feasible.

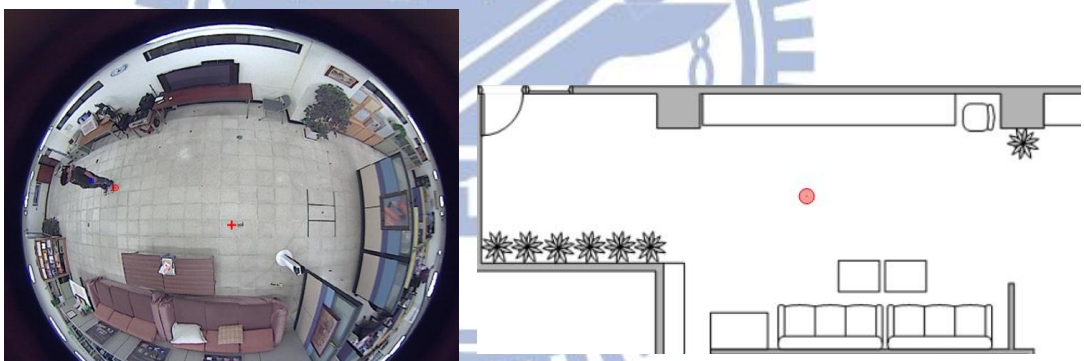
(a)



(b)



(c)



(d)

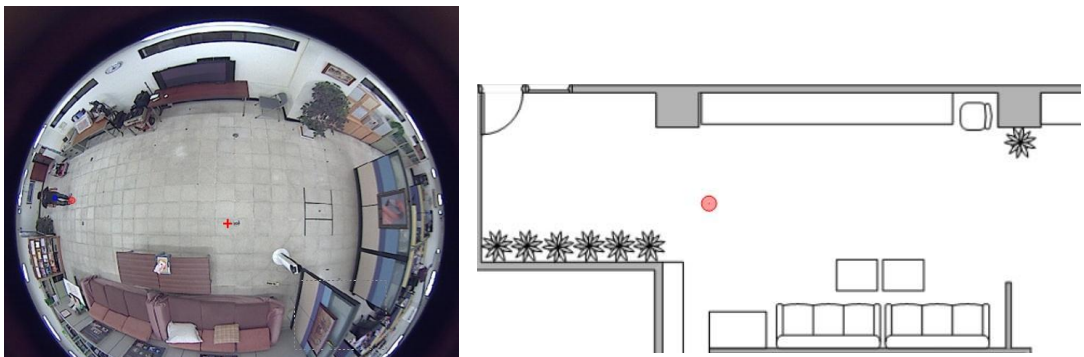
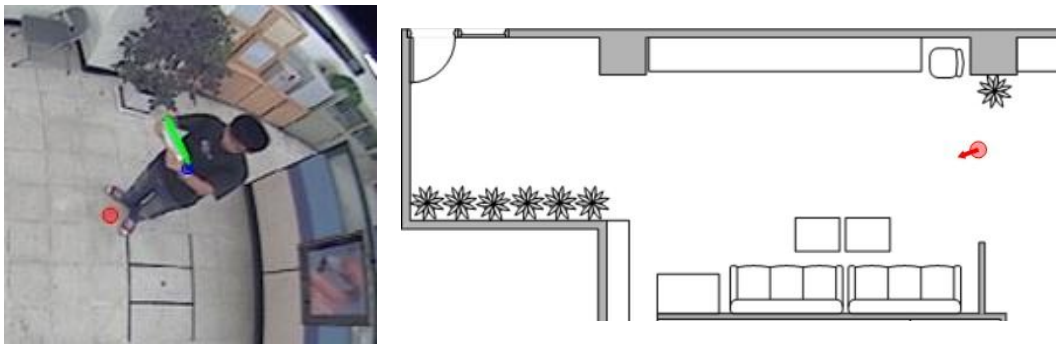
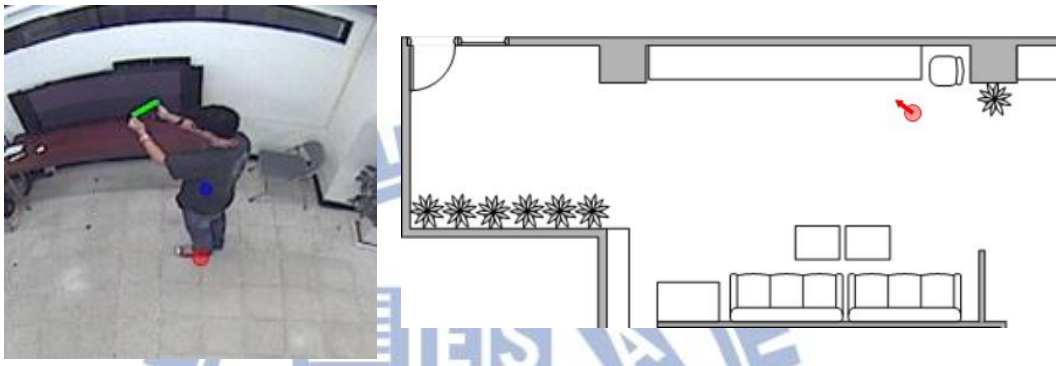


Figure 4.12 Human location detection at four different locations.

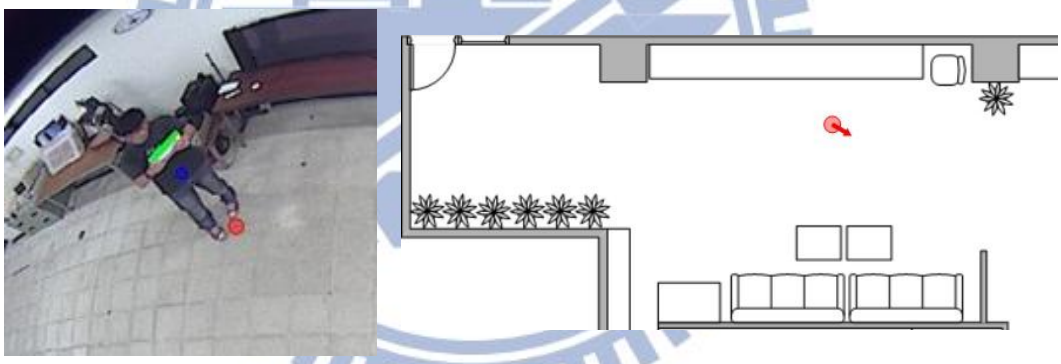
(a)



(b)



(c)



(d)

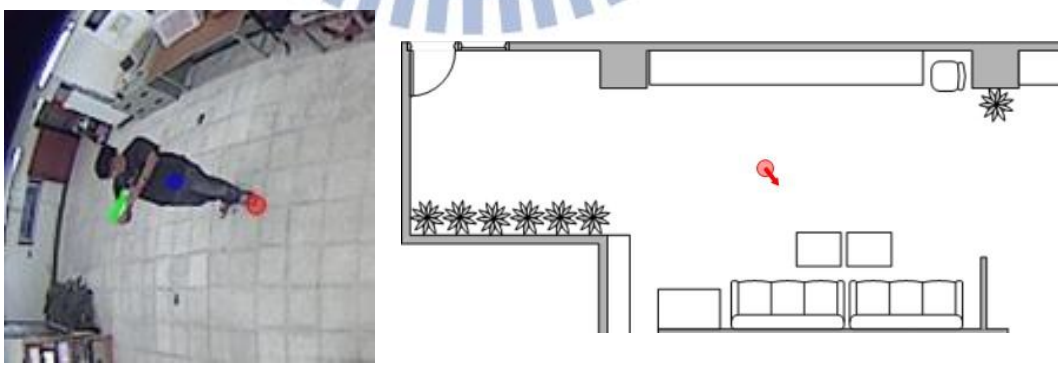


Figure 4.13 Human orientation detection by color edge mark at four different locations.

Chapter 5
Path Planning for Navigation

5.1 Ideas of Proposed Techniques

When a user want to reach a certain destination, the server-side system will find the location of the user at first, and then find the location of the destination in the environment map we constructed in the learning stage. Next, the server-side system will begin to plan a path starting from the user and ending at the destination, and sends the result to the client-side system.

Here we use an *obstacle image* obtained in the learning stage to determine whether a planned path collides with any obstacle or not. If the path starting from the location of a user and ending at the location of the destination does not collide with any obstacle, the server-side system directly sends the two locations to the client-side system, which means that the user may now walk forward to the desired destination. However, if the path collides with obstacles, we have to determine the immediate collision points to avoid the obstacles. Here we use an obstacle avoidance map constructed in the learning stage for this purpose. A more detailed description of the obstacle avoidance process will be described in Section 5.2. Next, we follow the avoidance directions to find the immediate points, and finally we will obtain a new path starting from the user's location to the destination. The path finding scheme will be described in Section 5.3. However, the planned path may not be in the simplest form; in other words, there may exist two non-connected points on the path that can instead be connected together without any obstacle between the two points. Therefore,

we propose a scheme to simplify the planned plan, which will be described in Section 5.4.

After a path is completely planned, it will be send to the client-side system. A user can follow the path to reach the desired destination. However, if the user moves to a location which is not on the planned path, the planned path must be updated. A path update scheme is also proposed in this study, which will be described in Section 5.5.

Finally, we will describe a complete path planning process in Section 5.6. Then some experimental results will be presented in the last section.

5.2 Obstacle Avoidance

An obstacle avoidance map is created by splitting the MCS into small blocks. A block is a processing unit in the obstacle avoidance process; in other words, the planned path "walks" a block at a time, if the planned path walks to a block containing obstacles, it will find another block to go. Here we introduce a *block coordinate space* (BCS), which can be used to locate the position of a block. The BCS coordinates of a block are denoted as (i, j). A block with the BCS coordinates (0, 0) means that the block is at the top-left corner; a block with the BCS coordinates (1, 0) means that it is the one on the right side of the one with coordinates (0, 0), and so on. The MCS coordinates (M_x, M_y) of a block with the BCS coordinates (i, j) can be computed by the following equations:

$$\begin{aligned} M_x &= n \cdot (i + 0.5); \\ M_y &= n \cdot (j + 0.5), \end{aligned} \tag{5.1}$$

where n is the size of a block in pixels. And the BCS coordinates (i, j) can also be computed by the following equations:

$$i = \left\lfloor \frac{M_x}{n} \right\rfloor;$$
$$j = \left\lfloor \frac{M_y}{n} \right\rfloor. \tag{5.2}$$

Each element in an obstacle avoidance map represents two opposite avoidance directions, so we only store the one of the two directions in degrees. As shown in Figure 5.1, the region of all angle degrees can be divided to 8 parts all with an equal degree range of 45^o. Every region part is assigned an index from 0 to 7, where the degree region of part 0 is from 337.5^o to 22.5^o; the degree region of part 1 is from 22.5^o to 67.5^o, and so on. Therefore, we can determine the region part of an avoidance direction by the angle of the direction. The region part of an avoidance direction is called "*avoidance region"* in the sequel.

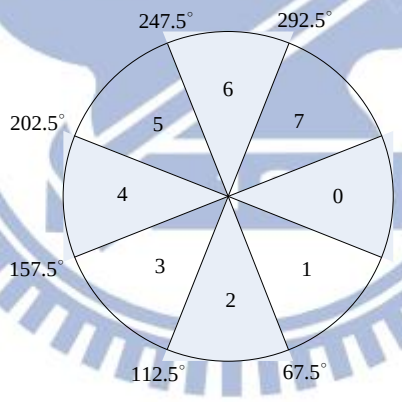


Figure 5.1 The whole direction region is divided to 8 parts, and each part is assigned an index.

As the 3 × 3 blocks shown in Figure 5.2, if a planned path walks to the central block and cannot directly walk to the destination from the block, the proposed system tries to find the next immediate block in the 7 neighborhoods by the avoidance direction of the block. Here, we can apply the avoidance regions to the 3 × 3 blocks, and then assign each of the 7 neighborhoods an index as shown in Figure 5.2.

Therefore, each avoidance direction can map to an *avoidance block* by the index of the avoidance region. However, we do not find the immediate point just at one avoidance block. More specifically, we will assign three avoidance blocks for each avoidance range. As shown in Figure 5.3, each avoidance range (shown as semi-transparent regions) is assigned three blocks, which include one primary avoidance block (shown as red regions) of the same avoidance range and the two neighborhoods, which are called *secondary* avoidance blocks (shown as blue regions).

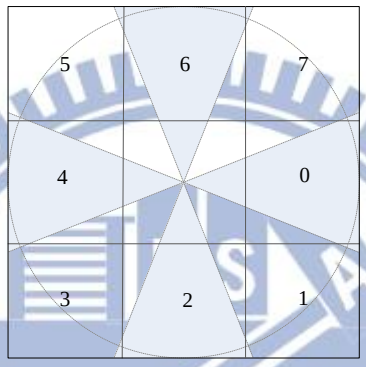


Figure 5.2 Apply the direction region parts to the neighborhoods of one block, and each neighborhood is assigned an index.

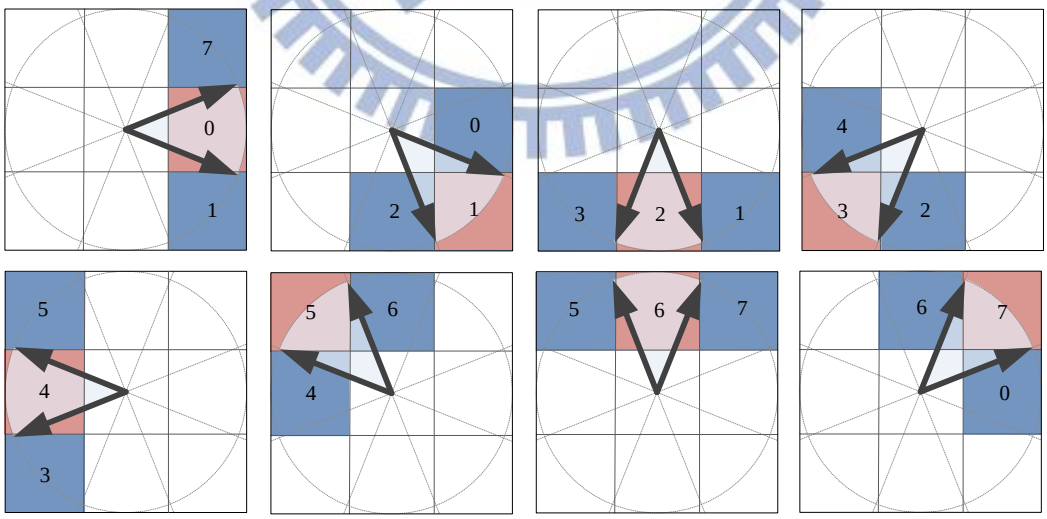


Figure 5.3 Avoidance blocks of 8 avoidance ranges, where the avoidance regions are shown as semi-transparent regions. Each avoidance region is assigned three blocks, which include the primary avoidance block (shown as red regions) of the same avoidance range and two secondary avoidance blocks (shown as blue regions).

We can find the next immediate point in the three avoidance blocks by the avoidance direction. More specifically, we have two avoidance directions in one block, so we will have six avoidance blocks for one block.

The following algorithm describes the processes to find the avoidance points of a block. It results in a set of avoidance points, which is sorted by the priority of the avoidance points. The priority of an avoidance point is based on the distance from the avoidance point to the destination point. An avoidance point with a higher priority should be considered first as the next immediate point in the path finding process. The path finding process scheme will be described in the next section.

Algorithm 5.1 Finding avoidance points.

Input: The current position p of a planned path in the MCS, the final destination position d of the planned path in the MCS, an obstacle image I_o, and an obstacle avoidance map A, where $A(i, j)$ means the angle of the avoidance direction of the block with BCS coordinates (i, j).

Output: A set of avoidance points, which is sorted by the priority of the avoidance block.

Steps

Step 1. Initialize an empty set S_{result} for the resulting points.

Step 2. Compute the coordinates (i, j) of p in the BCS by Equation 5.2.

Step 3. If $A(i, j)$ is non-negative, obtain the two avoidance directions $\overrightarrow{d_1}$ and $\overrightarrow{d_2}$ from $A(i, j)$; otherwise, take the following steps.

3.1. Compute the direction $\vec{v} = \overrightarrow{pd}$.

3.2. Find three avoidance blocks of the avoidance direction $\vec{v}$.

3.3. Go to Step 5.

Step 4. Find six avoidance blocks *B* of $\overrightarrow{d_1}$ and $\overrightarrow{d_2}$, which include two primary avoidance blocks and four secondary blocks.

Step 5. Initialize two empty ordered sets S_p and S_s.

Step 6. Take the following steps for each avoidance block *b* in *B*.

6.1. Compute the MCS coordinates b' of *b* by Equation 5.1.

6.2. If the line segment $\overline{pb'}$ collides with any obstacle, skip to the next avoidance block *b* and go to Step 6 again.

6.3. If *b* is of a primary avoidance block, add b' into S_p; otherwise, add b' into S_s.

Step 7. If S_p is not empty, take the following steps.

7.1. Sort each element b' of S_p in the ascending order by the *Manhattan distance* from b' to *d*:

$$|b'_x - d_x| + |b'_y - d_y|.$$

7.2. Add the first element of S_p into S_{result}, and add the remainder into S_s.

Step 8. Sort S_s by the same scheme of 7.1.

Step 9. Add each element of S_s into S_{result} orderly.

Step 10. Take S_{result} as the output.

In Step 3, if the current block contains no avoidance direction, the avoidance direction is set to the one from the current position to the destination. We add each of the avoidance blocks which are reachable from the current block into two sets. The primary avoidance blocks will be added into one set, and the secondary will be added

into another. Then we sort the sets based on the distance to the final destination in Step 7 and Step 8. We choose the closest one of the primary avoidance block as the first priority avoidance block, and add it into a resulting set. Finally, the remainders are added into the resulting set orderly.

5.3 Path Finding

We introduce the processes of obstacle avoidance in the previous section. In this section, we describe the path finding scheme by use of avoidance points.

When we want to find a path from the current position to a destination position, we find the avoidance points of the current position by Algorithm 5.1 at first. Then we begin to check each avoidance point. We will record it when we check an avoidance point, and we do not check the same avoidance point twice, which means that a path do not check a point it has walked before. If an avoidance point has not been walked before, we try to find a path from the avoidance point to the destination. Therefore, the process can become recursive. However, if an avoidance point cannot reach the destination, we continue to try the next avoidance point. If there are no more avoidance points, we finish the finding process. The following algorithm always returns a path starting from the input start point and ending at the destination if a path is found; otherwise, it returns a flag indicating the failure.

Algorithm 5.2 Path finding.

Input: A start point p in the MCS, A final destination point d in the MCS, and a set S_w which includes the points we have walked before.

Output: A set of points of the found path S_{result}, and a flag f indicating whether a path is found or not.

Steps

Step 1. Initialize an empty set S_{result} to store the output points.

Step 2. Add p into S_{result} and S_w.

Step 3. If p can directly go to d without colliding obstacles, add d into S_{result} and then go to Step 9.

Step 4. Find the avoidance points by Algorithm 5.1, resulting in a set of avoidance points S_a.

Step 5. Take out the first avoidance point p_a in S_a.

Step 6. If p_a is contained in S_w, take the following steps.

6.1. If there still is any avoidance point in S_a, go to Step 5; otherwise, set f to *fail* and then finish this algorithm.

Step 7. Apply this algorithm recursively with the inputs p_a, d, and S_w, resulting in a set of points of a found path S_{ad} and a flag f_a.

Step 8. If flag f_a is a *success*, add each point of S_{ad} into S_{result}; otherwise, go to Step 5.

Step 9. Set S_{result} as output and set f to be a *success*.

Because the start point should always be walked and contained in the resulting path, so we add the start point into the resulting set and the walked point set. In Step 3, we check whether the start point can be directly connected to the destination point without colliding obstacles; if so, then return the path which contains only the start point and the destination point. Otherwise, we have to find a next immediate point by applying the same algorithm recursively.

In Step 8 we check the returning flag: if the path is found successfully, then we add all the points of the returning points into the resulting set; otherwise, we try the next avoidance point again.

A result of the path finding process is shown in Figure 5.4. We found a path starting from the top-left point to the bottom-right point. As mentioned previously, the path is not of the simplest form. We will describe the processes to simplify the path in next section.

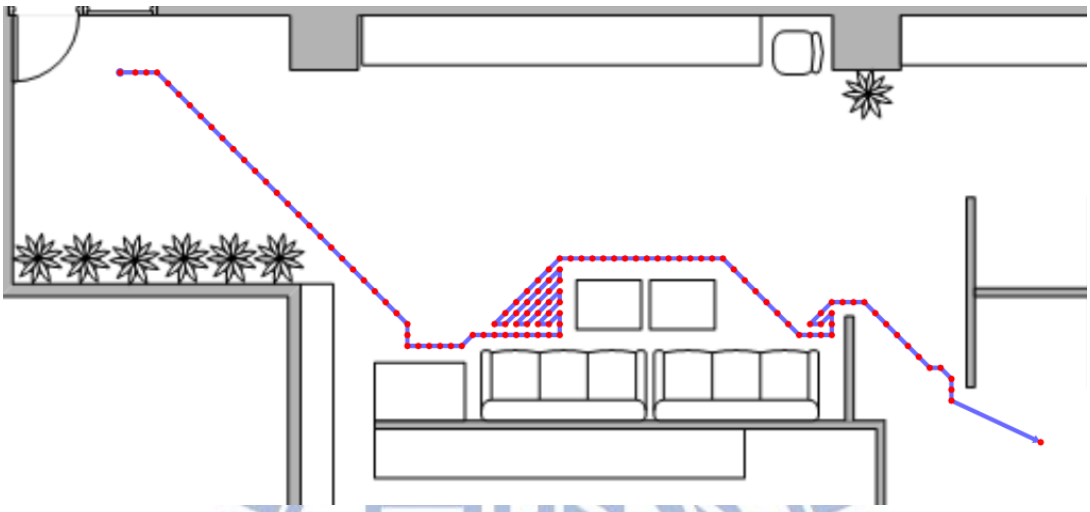


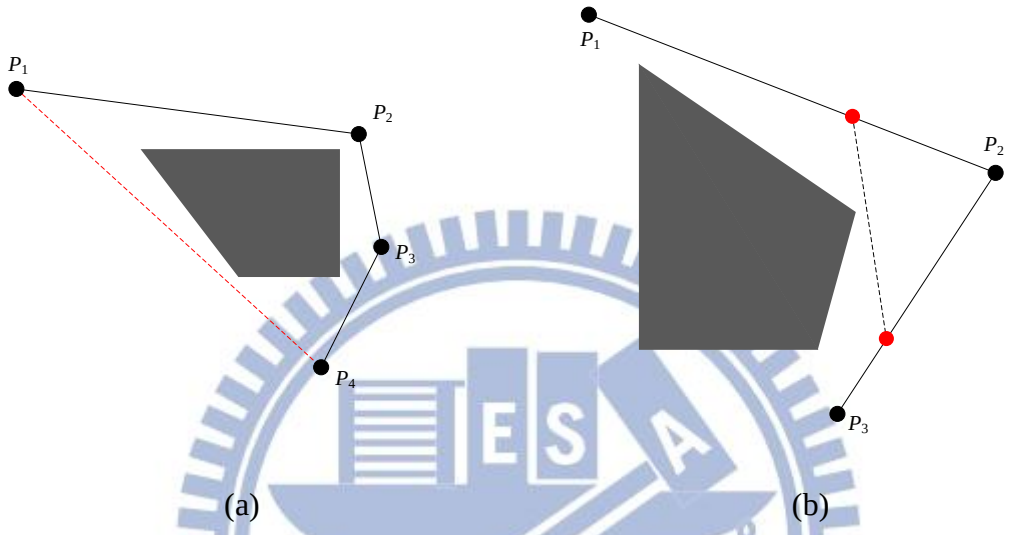
Figure 5.4 Path found in the path finding process.

5.4 Path Simplification

After a path is found, we begin to conduct the proposed path simplification process. The process can be decomposed into two parts: *redundant point elimination* and *distance elimination*. The goal of the *redundant point elimination* is to find two points which are non-connected and can instead be connected together, and then to remove the points between the two points; in other words, the goal is to find a "shortcut" between two points (like the red line shown in Figure 5.5(a)). The goal of *distance elimination* is to reduce the total path length by finding two points which are on different line segments and can be connected together; in other words, the goal is to find a "shortcut" between two line segments. As shown in Figure 5.5(b), we can reduce the total length of the path by substituting P_2 with two new immediate points

(shown as red points).

After the path finding process, we conduct redundant point elimination at first, as described in Algorithm 5.3. The algorithm iterates over all immediate points in a path, and finds the last immediate point which can be connected together for each immediate point.



(a) (b)

Figure 5.5 The redundant point elimination and the distance elimination. The black points represent the original immediate points of a path (a) The redundant point elimination, where the two redundant points P_2 and P_3 can be removed. (b) The distance elimination, the path length can be eliminated by substituting P_2 by the two red points.

Algorithm 5.3 Redundant point elimination.

Input: A set of points of a path *P*, where *P*(*i*) means the *i*th point in the set.

Output: A set of points of a simplified version of *P*.

Steps

Step 1. Initialize a variable $i = 1$ used to represent the index of the start point of the shortcut.

Step 2. If *i* is equal to *size*(*P*), go to Step 8, where *size*(*x*) means the number of the points of the set *x*.

Step 3. Initialize a variable $j = size(P)$ used to represent the index of the end point of the shortcut.

Step 4. If $i = j - 1$, go to Step 7.

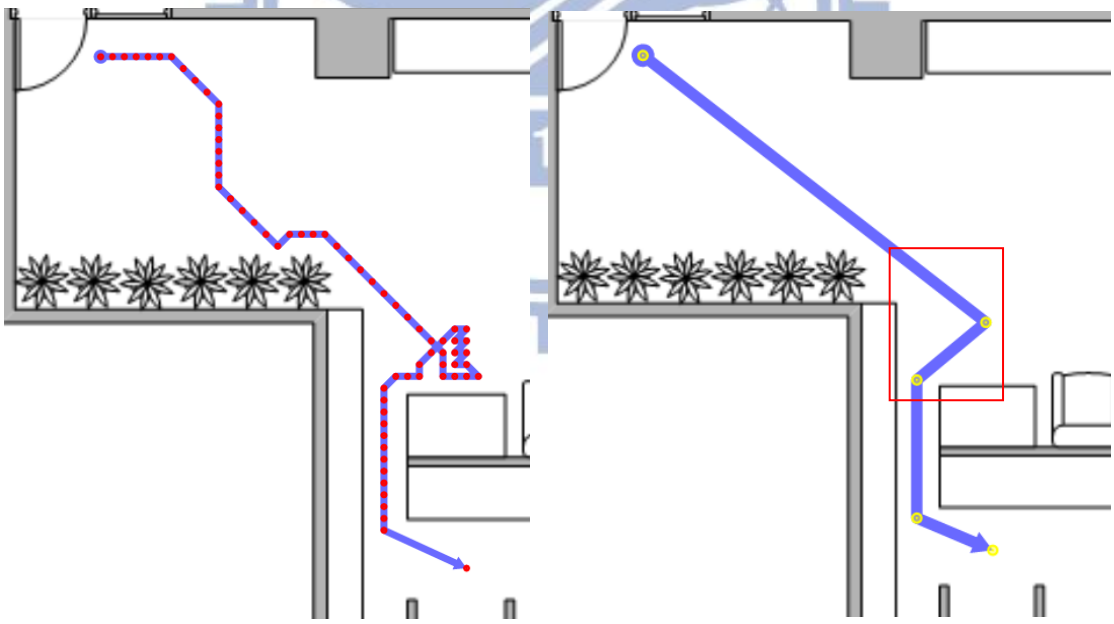
Step 5. If $P(i)$ can be directly connected to $P(j)$ without colliding obstacles, remove $P(i+1)$, $P(i+2)$, …, $P(j-1)$ and go to Step 7.

Step 6. Decrement j by 1 and go to Step 4.

Step 7. Increment i by 1 and go to Step 2.

Step 8. Take P as the output.

A result of redundant point elimination using the above algorithm is shown in Figure 5.6, where Figure 5.6(b) shows the result of applying redundant point elimination on Figure 5.6(a).



(a) (b)

Figure 5.6 Result of path finding and redundant point elimination. (a) Result of path finding. (b) Result of applying the redundant point elimination on (a).

However, the result shown in Figure 5.6(b) is still not of the simplest form. We

can find a short cut in the region of the red rectangle outline of Figure 5.6(b) by the distance elimination mentioned previously.

The distance elimination process checks the points on each line segment of a path. However, if the points on a line segment are continuous, it is impossible to check all points on the segment. Therefore, the points on a line segment are discretized into several points with equal distances T_d before distance elimination is conducted.

As shown in Figure 5.7, we check two line segments at a time, which are L_i and $L_{i+1,}$ respectively. For each discretized point p_i on L_i and each p_{i+1} on L_{i+1}, we check whether p_i and p_{i+1} can be connected together or not. The checking order of L_i is from the start point to the end point; that of L_{i+1} is contrarily from the end point to the start point. We check all discretized points on L_{i+1} for each p_i. If p_i (shown as blue points) cannot be connected to a point on L_{i+1}, it is skipped and the next p_i on L_{i+1} is then processed. Finally, we will find a shortcut between L_i and L_{i+1} if there exists one (shown as the red line segment).

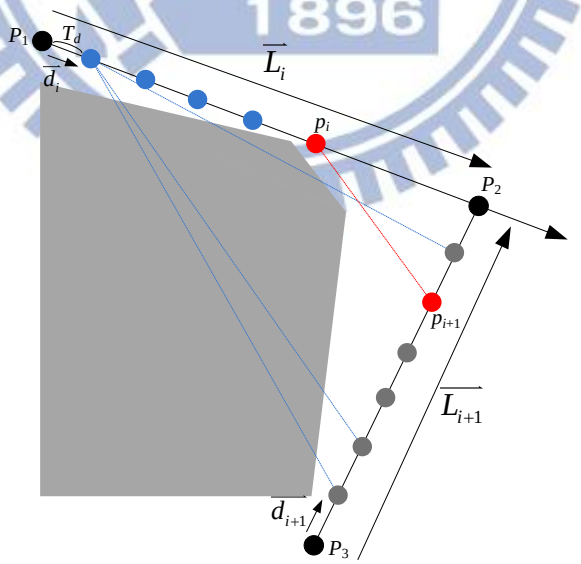


Figure 5.7 Process of distance elimination. The black points are the immediate points of a path. The gray region represents the region of an obstacle, the line between the two red points are a shortcut found by the distance elimination process.

The following algorithm describes the complete processes to perform the distance elimination work.

Algorithm 5.4 Distance elimination.

Input: A set of points of a path *P*, where *P*(*i*) means the *i*th point in the set.

Output: A set of points of a simplified version of *P*.

Steps

Step 1. Initialize a variable $i = 1$ used to represent the index of the first checking line segment.

Step 2. If $i \geq size(P) - 2$, regard that the last line segment has been reached, and go to Step 4; otherwise, take the following steps, where *size*(*x*) means the number of the points of the set *x*.

2.1 Initialize two vectors of the two checked line segments $\overrightarrow{L_i} = \overrightarrow{P(i)P(i+1)}$ and $\overrightarrow{L_{i+1}} = \overrightarrow{P(i+2)P(i+1)}$.

2.2 Compute the vectors $\overrightarrow{d_i}$ and $\overrightarrow{d_{i+1}}$ by the following equations:

$$\overrightarrow{d_i} = T_d \frac{\overrightarrow{L_i}}{\left|\overrightarrow{L_i}\right|}; \quad \overrightarrow{d_{i+1}} = T_d \frac{\overrightarrow{L_{i+1}}}{\left|\overrightarrow{L_{i+1}}\right|}$$

where T_d is a predefined distance between two neighboring discretized points.

2.3 Initialize two variables $p_i = P(i)$ and $p_{i+1} = P(i+2)$, which are used to represent the two end points of the shortcut, respectively.

2.4 Add $\overrightarrow{d_{i+1}}$ to p_{i+1}.

2.5 If $\overrightarrow{p_{i+1}P(i+1)}$ is in the opposite direction of L_{i+1} or $\left|\overrightarrow{p_{i+1}P(i+1)}\right| = 0$, then go to Step 3 to check the next two line segments.

2.6 If p_i can be connected directly to p_{i+1} without colliding obstacles, take

the following steps.

2.6.1. Insert p_i and p_{i+1} into P before the position of P(i+1).

2.6.2. Remove $P(i+1)$ from P.

2.6.3. Go to Step 3 to check the next two line segments.

2.7 Add $\overrightarrow{d_i}$ to p_i.

2.8 If $\left|\overrightarrow{p_i}\right| \geq \left|\overrightarrow{L_i}\right|$, go to Step 3; otherwise, go to 2.4.

Step 3. Increment i by 1 and go to Step 2.

Step 4. Take P as the output.

The result of applying the distance elimination algorithm described above on Figure 5.6(b) is shown in Figure 5.8. However, there still exist redundant points in the path of Figure 5.8. Therefore, we have to apply Algorithm 5.3 on the resulting path to reduce redundant points again. More specifically, we apply the redundant point elimination and distance elimination processes on a path until the points of the path are not changed. Algorithm 5.5 describes the above process.

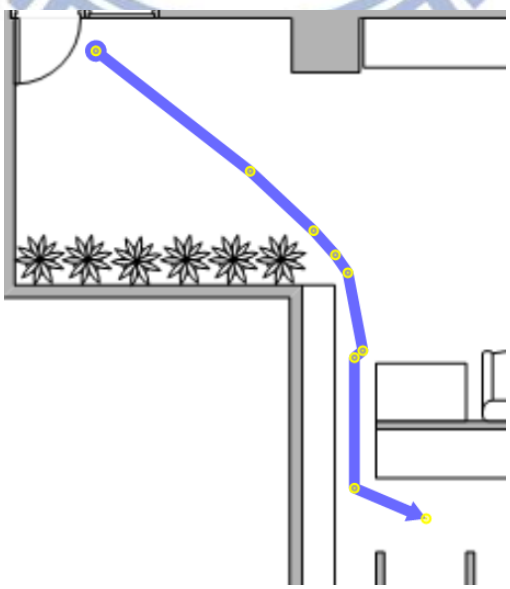


Figure 5.8 Result of applying the distance elimination on the path of Figure 5.6(b).

Algorithm 5.5 Path simplification.

Input: A set of points of a path P, where $P(i)$ means the ith point in the set.

Output: A set of points of the simplified path.

Steps

Step 1. Make a copy of P, and denote it by P'.

Step 2. Apply Algorithm 5.3 on P'.

Step 3. Apply Algorithm 5.4 on P'.

Step 4. If the points of P' are different than the points of P, clear P, copy all points of P' into P, and go to Step 2; otherwise, finish this algorithm.

We can directly apply Algorithm 5.5 on the resulting points of the path finding process. Figure 5.9 shows the result of applying the above path simplification process on the path of Figure 5.6(a).

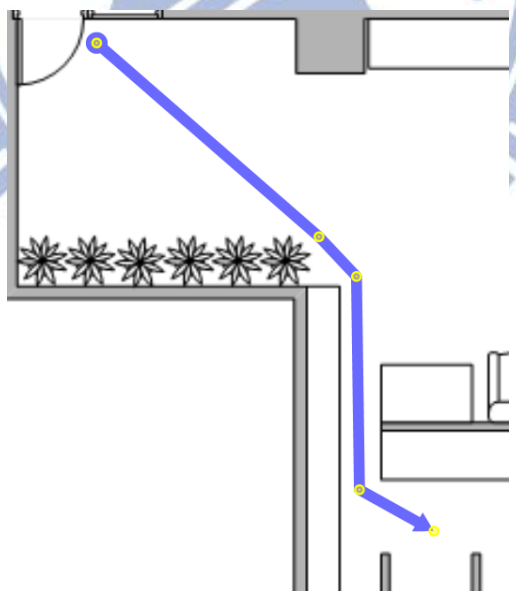


Figure 5.9 Result of applying the path simplification on the path of Figure 5.6(a).

5.5 Path Update

The set of the immediate points of a path will be sent to a client-side system. However, a user might not always move by following the planned path. Instead, the user might walk away from the planned path, so that the planned path becomes invalid. Therefore, we have to update a planned path when a user walks away from the planned path.

Algorithm 5.6 Path update.

Input: The current point p of a user, and a set of points of a planned path P, where $P(i)$ means the ith point in the set.

Output: A set of points of the updated path.

Steps

Step 1. Initialize an empty set P_{new}.

Step 2. Find the last point p_s of P which can be reached from p, if p_s is not found, take the following steps to re-plan a path.

2.1. Find a path starting from p and ending at the last point of P by Algorithm 5.2 and Algorithm 5.5, resulting in a set of points P'.

2.2. Add all points of P' into P_{new}, and go to 5.2.

Step 3. Add p into P_{new}.

Step 4. Add $P(i)$, $P(i+2)$, …, $P(n)$ into P_{new}, where i is the index of p_s in P and n is the size of P_{new}.

Step 5. If P_{new} contains at least 3 points, take the following steps.

5.1. Compute the angle θ between the two vectors $\overrightarrow{P_{new}(2)P_{new}(1)}$ and $\overrightarrow{P_{new}(2)P_{new}(3)}$ of the first two line segments of P_{new}.

5.2. If $\theta > \frac{\pi}{2}$, take the following steps to simplify the first two line segments of P_{new}.

5.2.1. on the first two line segments of P_{new}, resulting a set of points P_{start}.

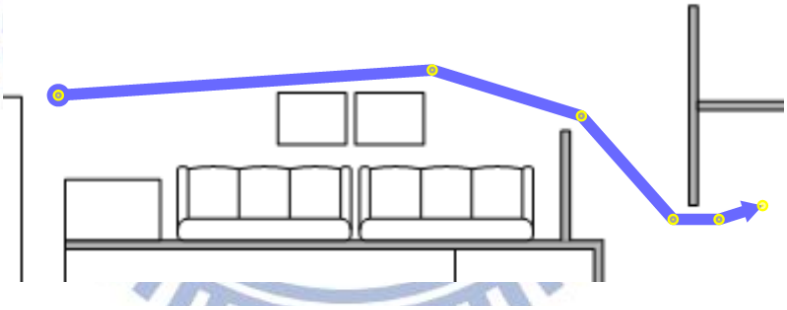
5.2.2. Remove the first two line segments from P_{new}.

5.2.3. Insert P_{start} to the beginning of P_{new}.

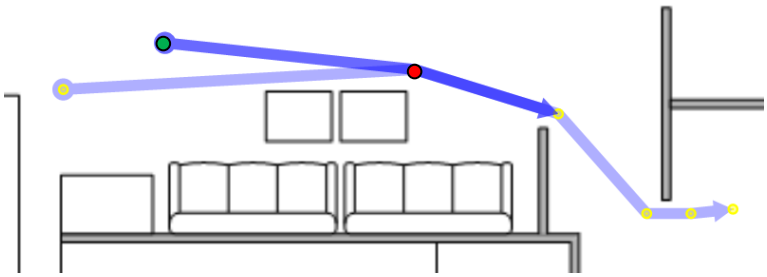
Step 6. Take P_{new} as the output.

An example of the results of applying the above algorithm is shown in Figure 5.10. When a user moves away from a planned path (as shown in Figure 5.10(a)), we find the last reachable point (shown as red circles) of the planned path from the current point (shown as green circles).

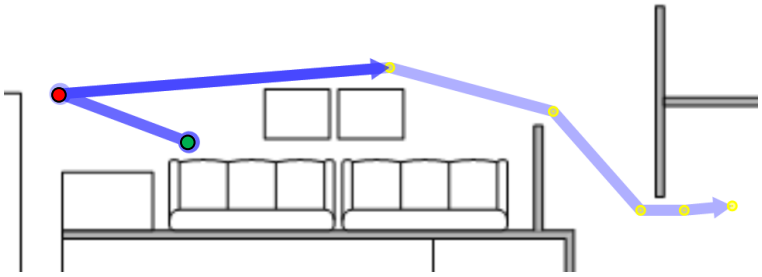
(a)



(b)



(c)



(d)

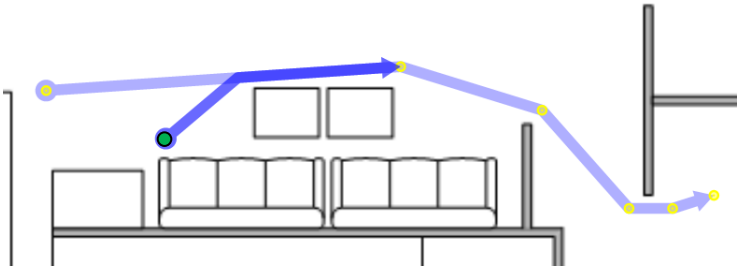


Figure 5.10 Results of the path update process. (green circles indicate the current point and red circles indicate the last reachable point from the current point) (a) The original planned path. (b) An updated path. (c) An updated path which is not of the simplest form. (d) Result of applying the path simplification on the path of (c).

After applying this algorithm, we will add at most one new point to the resulting set; in other words, the resulting set contains only one new point, which is the current point of the user, and the remainders are the points of the original planned path Therefore, we have to check whether the new line segment is of the simplest form or not. In Step 5.1, we compute the angle between the new line segment and the first old line segment. If the angle is greater than 90^{o} (as shown in Figure 5.10(c)), it means that the two line segments have opposite directions and there may exist a shortcut between the two line segments. Therefore, we apply the path simplification process to the two line segments (as shown in Figure 5.10(d)). Finally, the algorithm will yield an updated path.

5.6 Algorithm for Path Planning

We describe the processes for tasks involved in path planning in the previous sections. In this section, we integrate these processes to form a complete algorithm for path planning.

Algorithm 5.7 Path planning.

Input: A start point *p* in the MCS, and a final destination point *d* in the MCS.

Output: A set of points of the planned path.

Steps

Step 1. If there exists a planned path P' and the destination of P' is identical to d, take the following steps.

1.1. Update P' by Algorithm 5.6, resulting in a set P of the immediate points of a path.

1.2. Go to Step 4.

Step 2. Find a path starting from p and ending at d by Algorithm 5.2, resulting in a set P of immediate points of the path.

Step 3. Apply Algorithm 5.5 on P to simplify the found path.

Step 4. Take P as the output.

This algorithm is applied in each navigation process cycle. If a user wants to reach a certain destination and he/she never searched the same destination before, the algorithm will plan a new path. Otherwise, the algorithm will update the planned path.

5.7 Experimental Results

Figure 5.11, 5.12, and 5.13 show three examples of the results of the path planning work conducted by the above algorithm. Each result contains a figure of the original planned path yielded by Algorithm 5.2, and a figure of the final simplified path yielded by Algorithm 5.5.

(a)

(b)

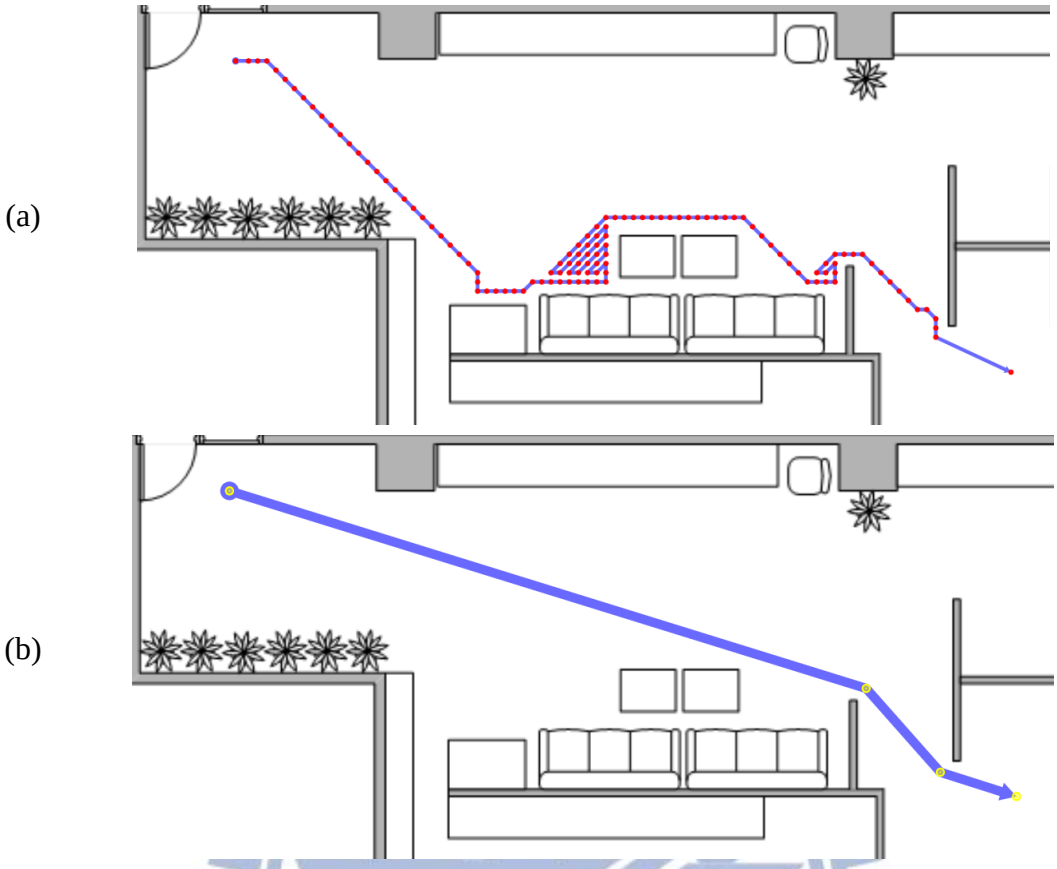
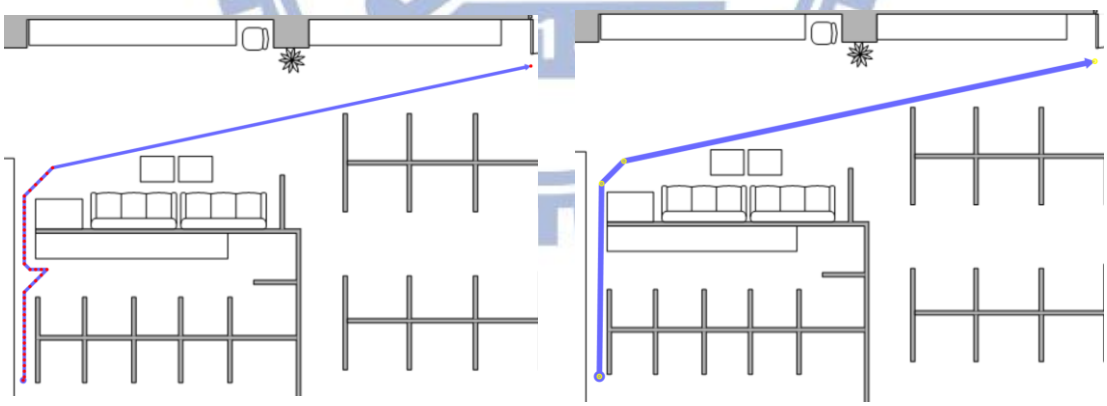
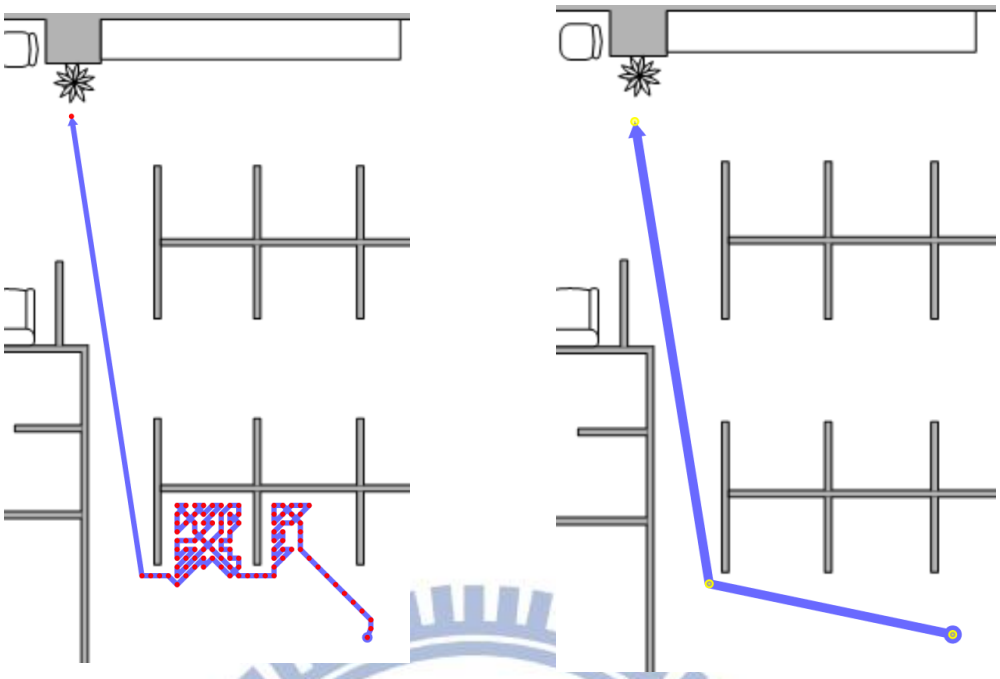


Figure 5.11 Result of the path planning. (a) Result of the path finding. (b) Result of applying the path simplification on the path of (a).



(a) (b)

Figure 5.12 Result of the path planning. (a) Result of the path finding. (b) Result of applying the path simplification on the path of (a).



(a) (b)

Figure 5.13 Result of the path planning. (a) Result of the path finding. (b) Result of applying the path simplification on the path of (a).

Chapter 6
Augmented Reality for Navigation

6.1 Ideas of Proposed Techniques

In this chapter, we will describe the AR techniques used in the proposed system. The AR techniques are used in the client-side system. We overlay navigation information onto the real images taken of the current scene, so that users can just take their mobile devices and conduct the navigation conveniently. The real images taken of the current scene will be called "*scene images*" in the subsequent sections, and scene images overlaid with navigation information will be called "*augmented images*."

After detections of the user's location and orientation are completed, the navigation information will be sent to the user's mobile device. The navigation information includes the visiting target information and the navigation path. The client-side system will display the information on the device screen. A more detailed description of the navigation information we use in the display rendering will be given in Section 6.2.1. The visiting target information includes the name of the visiting target and its coordinates in the GCS. The navigation path contains the GCS coordinates of the points consisting of the path. In order to display the information in an AR way, the client-side system must transform the GCS coordinates onto a 2D screen plane. The calibration of the camera on the mobile device is described in Chapter 3. In Section 6.2.2, we will describe the process to perform the transformation between the GCS and the screen plane by the calibration result.

In Section 6.3, we will describe the display rendering for the navigation

information. We display the names and distances of visiting targets on the corresponding objects in scene images. So we will illustrate how to determine the display position in the scene image in Section 6.3.1. In Section 6.3.2, we will describe the creation of the navigation path's geometric shape (arrows, thick line segments, etc.) for the navigation path to be overlaid onto the scene image to provide the guidance information.

6.2 View Mapping between Real World and Client Device

6.2.1 Information for Use in Mapping between Real World and Client Device

In Chapter 3, we described the construction of the environment map. We specify the visiting target information on the environment map, which includes the name, the region, and the coordinates of the visiting target. As shown in Figure 6.1, the light green region on the floor plan is a specified visiting target. The visiting target is specified by a vector $\vec{f}$ indicating the front direction of the visiting target, the region width *w* and height *h*, and the location *p*. The coordinates of the location include the *z*-coordinate, which represents the distance between the ground plane and the bottom of the visiting target region. The coordinates and the size are specified in the MCS, and it will be transformed into the GCS before sending to the client-side system.

Besides the visiting targets, a navigation path may be sent to the client-side system when a user wants to reach a certain destination. The navigation path is a set which contains the immediate points of the path, and the immediate points will be

transformed into the GCS before being sent to the client-side system. In addition, a path might contain more than one turning; in other words, a user might have to turn more than once to reach the destination. However, a user should pay attention only to the next turning; the second turning is not so important to the user at the current time. Therefore, we only display two line segments of the path at a time.

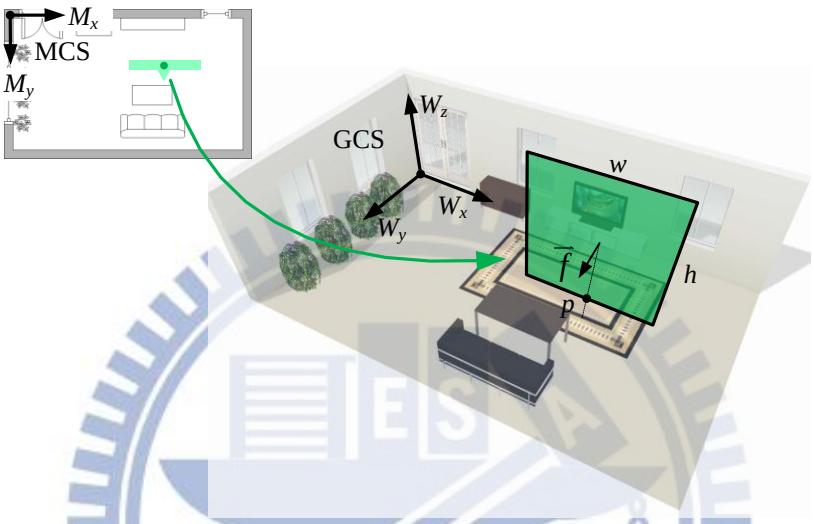


Figure 6.1 A visiting target in the environment map and its corresponding location in the GCS.

6.2.2 Transformation from Real World Spot to Client Device Screen

Recall the results derived from Section 3.4.2. A point p in the CCS can be transformed to be a point q in the ICS by the following equations:

$$\begin{pmatrix} p'_x \\ p'_y \\ p'_z \\ p'_w \end{pmatrix} = \begin{pmatrix} \frac{h}{w}\cot\frac{\alpha}{2} & 0 & 0 & 0 \\ 0 & \cot\frac{\alpha}{2} & 0 & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}; \qquad (6.1)$$

$$\begin{pmatrix} q_u \\ q_v \\ q_z \end{pmatrix} = \begin{pmatrix} w & 0 & 0 \\ 0 & h & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0.5 & 0 & 0 & 0.5 \\ 0 & -0.5 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0.5 \end{pmatrix} \cdot \begin{pmatrix} p'_x / p'_w \\ p'_y / p'_w \\ p'_z / p'_w \\ 1 \end{pmatrix}. \quad (6.2)$$

where w is the width of the scene image in the unit of pixel, h is the height, and α is the angle range of the field-of-view of the camera. The view region is restricted by the two parameters n and f; n restricts the smallest distance we can see, and f restricts the largest distance, so the two parameters can be specified arbitrarily. Equation 6.2 is a little different from the one in Section 3.4.2. The additional z-coordinate is only used to determine whether a point is outside of the screen range or not. A point with the z coordinate outside of the range [0, 1] is considered to be outside the screen range. The ICS coordinates are composed by (q_u, q_v).

However, the coordinates sent from the server-side is in the GCS, but the coordinates we use in Equation 6.1 are in the CCS. Therefore, we have to transform the coordinates from the GCS into the CCS at first. The transformation can be expressed by the following equation:

$$p = M_c \cdot \begin{pmatrix} a_x \\ a_y \\ a_z \\ 1 \end{pmatrix} \quad (6.3)$$

where a is a point in the GCS, p is the transformed point of a in the CCS, and M_c is the transformation matrix. In this transformation, the original coordinates of a are replaced by homogeneous coordinates.

Because the transformation will preserve the length of vectors after transformation, the transformation is so called *orthogonal transformation*. The columns of the transformation matrix of an orthogonal transformation will form an orthonormal basis of the transformed space. As shown in Figure 6.2(a), a camera is at the point c in the GCS, and the basis of the CCS of the camera can be represented by

three vectors *up*, *right*, and *forward*, where *up* is the up direction of the camera, *right* is the right direction, and *forward* is the front direction. In the CCS defined in Chapter 3 shown in Figure 6.2(b), we can express the transformation matrix as follows:

$$M_c = \begin{pmatrix} right_x & up_x & -forward_x & -c_x \\ right_y & up_y & -forward_y & -c_y \\ right_z & up_z & -forward_z & -c_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{6.4}$$

where the rightmost column is the coordinates of the camera, it used to translate the origin to the camera position, and it is actually the user's location detected from the server-side system. The z-coordinate of the camera position is a predefined parameter; in other words, the height of a camera is fixed to about the height of the eyes of an adult in the proposed system.

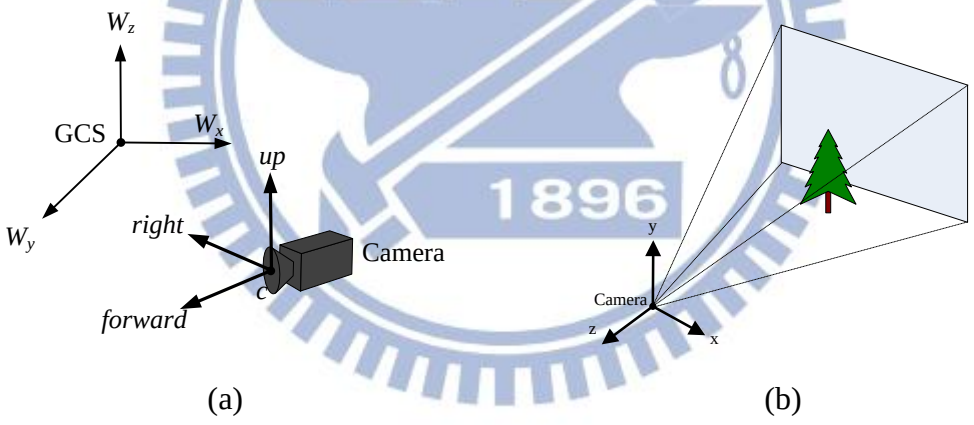


Figure 6.2 A camera in the GCS and the CCS. (a) A camera in the GCS with three orthonormal vectors *up*, *right*, and *forward*. (b) The CCS.

Therefore, we can determine the transformation matrix M_c by finding the three vectors *up*, *right*, and *forward*. By the GCS defined in Chapter 3, the up direction is the +z direction, so the vector *up* is (0, 0, 1).

Also, we assume that the camera orientation is in the same direction of the user's orientation. Therefore, the vector *forward* can be obtained by using the user's

orientation. However, a user might tilt the client device to watch the environment at a *pitch angle*. As shown in Figure 6.3, the camera looks at a pitch angle β, but the orientation detected from the server-side is from a horizontal direction (shown as the green arrow). Therefore, we have to obtain the pitch angle so that we can obtain the correct orientation by Equation 6.5 below:

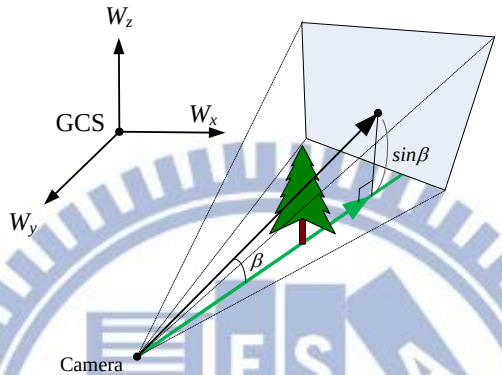


Figure 6.3 Camera looks at a pitch angle β. The green line indicates a line on the horizontal plane.

$$\vec{d}' = \begin{pmatrix} \vec{d}_x \\ \vec{d}_y \\ \sin\beta \end{pmatrix} \tag{6.5}$$

where $\vec{d}$ is the user's orientation detected from the server-side system, and β is the pitch angle which can be obtained from the orientation sensor of the client device. And then *forward* can be obtained by:

$$forward = \frac{\vec{d}'}{\left|\vec{d}'\right|}. \tag{6.6}$$

The value *forward* is normalized to be a unit vector in the equation above. The last vector *right* can then be obtained by the vector which is orthogonal to *up* and

forward. We can get the vector by the cross product of *forward* and *up* according to the following equations.

$$\vec{r} = forward \otimes up;$$
$$right = \frac{\vec{r}}{|\vec{r}|}. \tag{6.7}$$

However, *forward* might not be orthogonal to *up* which is the direction (0, 0, 1) because we had added a *z*-direction to the vector *forward*. Therefore, we correct the *up* by the cross product of *right* and *forward*:

$$up = right \otimes forward \tag{6.8}$$

Now, we have obtained all the needed variables to perform the transformation. We summarize all the processes of transformations discussed above by the following algorithm.

Algorithm 6.1 Transformation between the GCS and the ICS.

Input: A user's orientation $\vec{d}$, the user's location *c*, the pitch angle β obtained from the orientation sensor of the client device, and the point *a* in the GCS to transform.

Output: A transformed point *q* in the ICS.

Steps

Step 1. Initialize a vector *up* with the direction (0, 0, 1).

Step 2. Use $\vec{d}$ and β to compute the vector $\vec{d}'$ by Equation 6.5.

Step 3. Use $\vec{d}'$ to compute the vector *forward* by Equation 6.6.

Step 4. Compute a vector *right* by Equation 6.7.

Step 5. Correct a vector *up* by Equation 6.8.

Step 6. Construct the matrix M_c using *up*, *forward*, *right*, and *c* by the matrix of (6.4).

Step 7. Use M_c to transform a from the GCS to the CCS by Equation 6.3 and result in p.

Step 8. Transform p from the CCS to the ICS by Equations 6.1 and 6.2.

6.3 Rendering for Visiting Targets and Navigation Paths

In this section, we describe the schemes we propose to display visiting targets and navigation paths on the device screen.

6.3.1 Visiting Target Rendering

As shown in Figure 6.4, we try to overlay the name and the distance of visiting targets onto the corresponding objects in the real world, which appear in the image taken with the camera on the user-held mobile device. In order to accomplish this aim, we have to determine where to display the text on the device screen.

As shown in Figure 6.5, the visiting target is defined by four parameters which are described in the previous section. Then, the four points of the region of a visiting target can be computed by the following equations:

Figure 6.4 An augmented image overlaid with visiting target information.

$$
\begin{aligned}
&\overrightarrow{f'} = \frac{w}{2}(\overrightarrow{f}_y, -\overrightarrow{f}_x);\\
&a = (p_x - \overrightarrow{f'}_x,\ p_y - \overrightarrow{f'}_y,\ p_z + h);\\
&b = (p_x + \overrightarrow{f'}_x,\ p_y + \overrightarrow{f'}_y,\ p_z + h);\\
&c = (p_x + \overrightarrow{f'},\ p_y + \overrightarrow{f'}_y,\ p_z);\\
&d = (p_x - \overrightarrow{f'}_x,\ p_y - \overrightarrow{f'}_y,\ p_z).
\end{aligned}
\tag{6.9}
$$

Then we can compute the ICS coordinates of *a*, *b*, *c*, and *d* by Algorithm 6.1, resulting in four new values a', b', c', and d' as shown in Figure 6.6(a).

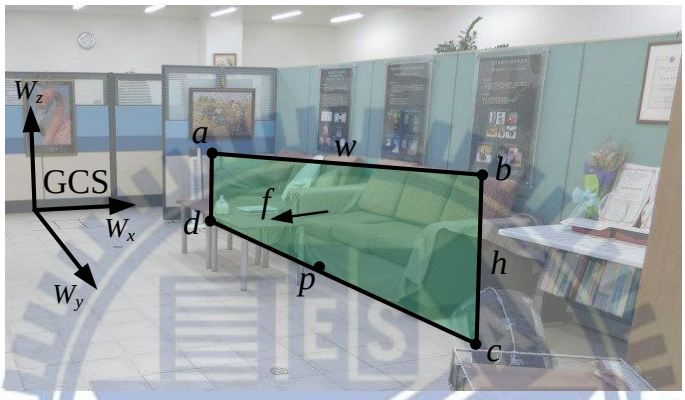
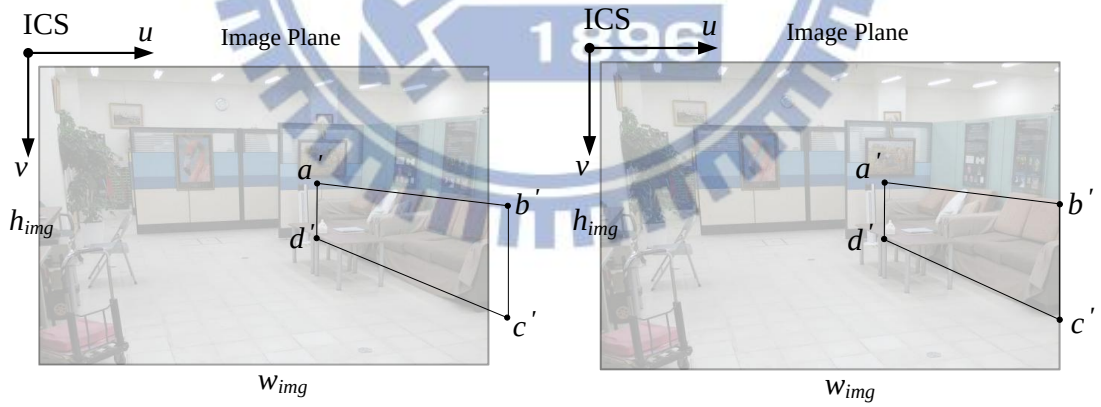


Figure 6.5 Parameters of a visiting target (shown as the green region). All the parameters are in the GCS.



(a) (b)

Figure 6.6 Four points transformed from the GCS of a visiting target. (a) Before clipping to the range of the image size. (b) After clipping to the range of the image size.

However, the transformed points might exceed the region of the scene image. Therefore, we clip the points by the following equation:

$$\begin{pmatrix} p'_u \\ p'_v \end{pmatrix} = \begin{pmatrix} \max\left(\min\left(p_u, w_{img}+1\right), -1\right) \\ \max\left(\min\left(p_v, h_{img}+1\right), -1\right) \end{pmatrix} \tag{6.10}$$

where p is the point to clip, p' is the point after clipping, w_{img} is the width of the scene image, and h_{img} is the height. The range of u coordinates is between $[-1, w_{img}+1]$ and v is between $[-1, h_{img}+1]$. We can determine whether a point p' is inside the screen region or not by checking $p'_u \in [0, w_{img}]$ and $p'_v \in [0, h_{img}]$. Then, we can obtain the display position of the text of the visiting target information by the following equation:

$$p_{text} = \frac{a' + b' + c' + d'}{4} \tag{6.11}$$

And then we can display the text on the display position p_{text} as shown in Figure 6.7, where w_{text} is the width of the text, and h_{text} is the height.

Figure 6.7 Display the visiting target information on the display position p_{text}.

Furthermore, when a user has searched a destination, he/she must know which direction to go. We want to let the user always know the direction of the destination target even the target is outside the range of screen. As shown in Figure 6.8, if a

destination target is outside the range of screen, we should display the visiting target on the edge of the screen (shown as the red point). Then, the user can understand what the direction the destination is in.

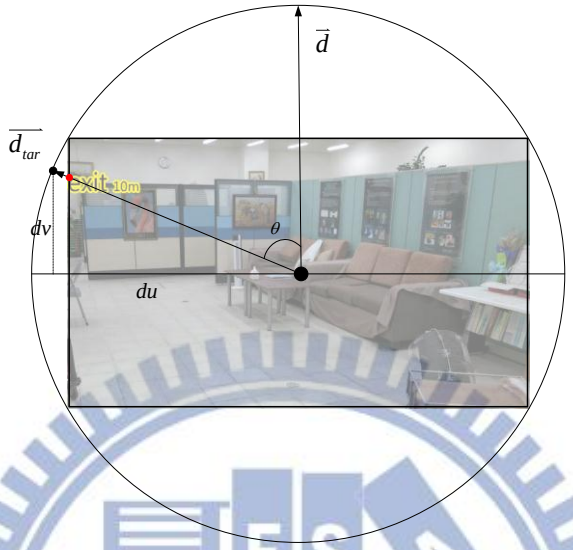


Figure 6.8 Display point for a visiting target which is outside of the screen range. $\vec{d}$ is the orientation of the user, and $\overrightarrow{d_{tar}}$ is the vector from the user's location to the visiting target.

We summarize all the processes of visiting target display rendering discussed above by the following algorithm.

Algorithm 6.2 Display rendering of a visiting target.

Input: An image *I* which is to be drawn, with the width w_{img} and the height h_{img}; a user's location *e*, and his/her orientation $\vec{d}$; a visiting target *tar* with a vector $\overrightarrow{f}$ indicating the front direction of the visiting target, the region with width *w* and height *h*, and its location *p*; a text *t* to display with the width w_{text} and the height h_{text}.

Output: An augmented image.

Steps

Step 1. Compute four points *a*, *b*, *c*, and *d* by Equation 6.9.

Step 2. Transform *a*, *b*, *c*, and *d* from the GCS into the ICS by Algorithm 6.1, resulting in a', b', c', and d'.

Step 3. Clip a', b', c', and d' into the range of the size of *I* by Equation 6.10.

Step 4. Compute the display position p_{text} by Equation 6.11.

Step 5. If $p_{text_u} \notin [0, w_{img}]$ or $p_{text_v} \notin [0, h_{img}]$ or $p_{text_z} \notin [0,1]$, take the following steps.

5.1 If *tar* is not the destination target of the user, finish this algorithm.

5.2 Compute the direction vector of *tar* by $\overrightarrow{d_{tar}} = \overrightarrow{ep} / \left|\overrightarrow{ep}\right|$ and the angle θ between $\overrightarrow{d_{tar}}$ and $\vec{d}$ (as shown in Figure 6.8).

5.3 Compute the diagonal length of the screen $d_{dia} = \sqrt{{w_{img}}^2 + {h_{img}}^2}$.

5.4 Compute *du* and *dv* by the following equations:

$$du = \frac{d_{dia}}{2}\cos(\theta - \frac{\pi}{2});$$

$$dv = \frac{d_{dia}}{2}\sin(\theta - \frac{\pi}{2}).$$

5.5 Compute p_{text} by the following equations if $\frac{-h_{img}}{2} < dv < \frac{h_{img}}{2}$:

$$p_{text_v} = \frac{h_{img}}{2} + \text{sign}(dv)\frac{w_{img}}{2}\left|\tan(\theta - \frac{\pi}{2})\right|;$$

$$p_{text_u} = w_{img}\left(\frac{\text{sign}(du)}{2} + 0.5\right).$$

where the function sign(*x*) returns −1 if *x* is negative; otherwise, it returns 1. If $\frac{-w_{img}}{2} < du < \frac{w_{img}}{2}$, then compute:

$$p_{text_u} = \frac{w_{img}}{2} + \text{sign}(du)\frac{h_{img}}{2}\frac{1}{\left|\tan(\theta - \pi/2)\right|};$$

$$p_{text_v} = h_{img}\left(\frac{\text{sign}(dv)}{2} + 0.5\right);$$

Otherwise compute:

$$p_{text_u} = \frac{w_{img}}{2} + du;$$

$$p_{text_v} = \frac{h_{img}}{2} + dv.$$

Step 6. Correct p_{text} to make the drawn text not exceeding the range of I by the following equation:

$$\begin{pmatrix} p'_{text_x} \\ p'_{text_y} \end{pmatrix} = \begin{pmatrix} \max\left(\min\left(p_{text_x}, w_{img} - \frac{w_{text}}{2} \right), \frac{w_{text}}{2} \right) \\ \max\left(\min\left(p_{text_y}, h_{img} - \frac{h_{text}}{2} \right), \frac{h_{text}}{2} \right) \end{pmatrix}.$$

Step 7. Draw t on I at the point p'_{text}.

Step 8. Take I as the output.

As mentioned previously, if a user's destination is outside the screen range, we display it on the edge of the screen. Therefore, we compute the appropriate display point (as shown by the red point in Figure 6.8) in the sub-steps of Step 5.

6.3.2 Rendering and Geometry Creation of Navigation Paths

As mentioned previously, only two line segments of a path will be displayed at a time. If a user receives a path sent from the server-side system as shown in Figure 6.9(a), then the first two line segments are what we are going to display. The path composed of the first two line segments will be called "*display path*" in the subsequent sections. Figure 6.9(b) shows the expected result of overlaying the display path onto a scene image. The display path in Figure 6.9(b) is composed of thick line segments and an arrow. Therefore, we have to create the geometric shape of a path before conducting display rendering for them.

A display path is a 3D augmented object. As mentioned in Chapter 2, the

rendering of 3D augmented objects is conducted by the OpenGL API. The OpenGL API processes 3D objects composed by triangles or quadrangles, which are called "*geometric primitives*." As shown in Figure 6.10, the geometric shape of a display path is composed by 11 points, and we can get the geometric primitives of the display path by the 11 points of a path. Accordingly, we can perform the display rendering for a navigation path by Algorithm 6.3.

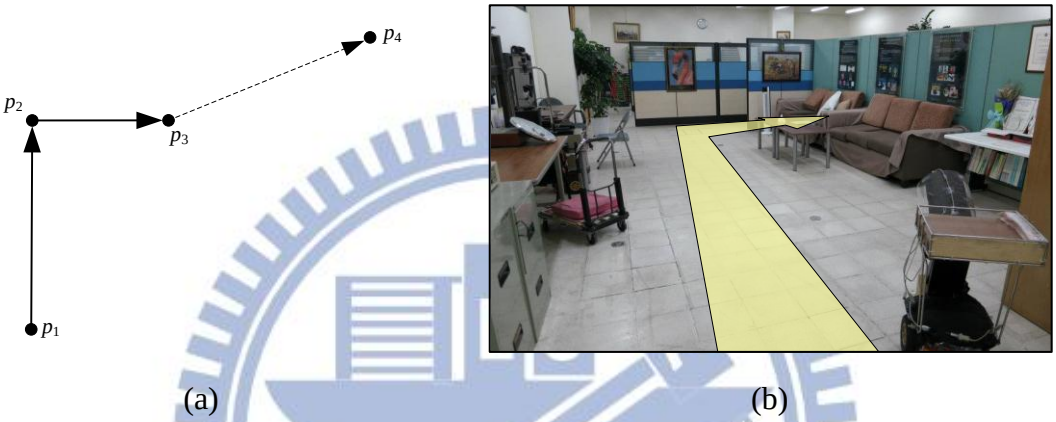


Figure 6.9 A path and its display on a screen. (a) The path with three line segments. The first two line segments are which should be concerned by a user. (b) The display of the first two line segments of the path of (a).

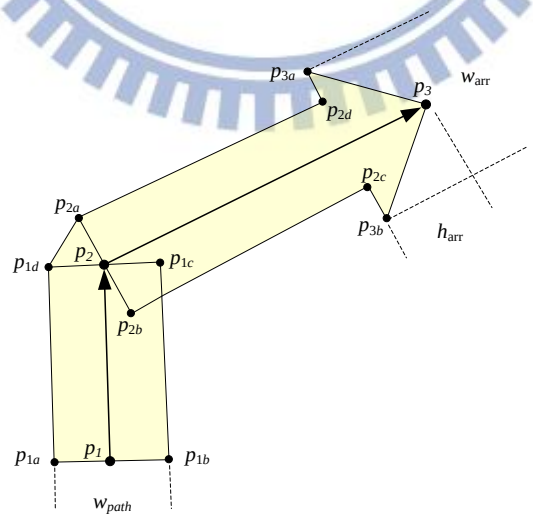


Figure 6.10 The geometry of a display path

Algorithm 6.3 Display rendering for a navigation path.

Input: A set P of points of a planned path, an image I to draw, the width w_{path} of the display path, the width w_{arr} of the arrow of the display path, and the length h_{arr} of the arrow.

Output: An augmented image.

Steps

Step 1. Take the first three points p_1, p_2, and p_3 of P.

Step 2. Compute $\vec{v}_1 = \frac{\overrightarrow{p_1p_2}}{\left|\overrightarrow{p_1p_2}\right|}$, $\vec{v}_2 = \frac{\overrightarrow{p_2p_3}}{\left|\overrightarrow{p_2p_3}\right|}$, $\vec{v}_1' = \frac{w_{path}}{2}(\vec{v}_{1y}, \overrightarrow{-v_{1x}}, 0)$, and $\vec{v}_2' = \frac{w_{path}}{2}(\vec{v}_{2y}, \overrightarrow{-v_{2x}}, 0)$.

Step 3. Compute $p_{1a}, p_{1b}, p_{1c}, p_{1d}$ by the following equations:

$$p_{1a} = p_1 + \vec{v}_1';$$
$$p_{1b} = p_1 - \vec{v}_1';$$
$$p_{1c} = p_2 - \vec{v}_1';$$
$$p_{1d} = p_2 + \vec{v}_1'.$$

Step 4. If $\left|\overrightarrow{p_2p_3}\right| < h_{arr}$, then set h_{arr} to $\left|\overrightarrow{p_2p_3}\right|$.

Step 5. Compute $p_{2a}, p_{2b}, p_{2c}, p_{2d}$ by the following equations:

$$p_{2a} = p_2 + \vec{v}_2';$$
$$p_{2b} = p_2 - \vec{v}_2';$$
$$p_{2c} = p_3 - h_{arr}\vec{v}_2 - \vec{v}_2';$$
$$p_{2d} = p_3 - h_{arr}\vec{v}_2 + \vec{v}_2'.$$

Step 6. Compute p_{3a}, p_{3b} by the following equations:

$$p_{3a} = p_3 - h_{arr}\vec{v}_2 + \frac{w_{arr}}{w_{path}}\vec{v}_2';$$
$$p_{3b} = p_3 - h_{arr}\vec{v}_2 - \frac{w_{arr}}{w_{path}}\vec{v}_2'.$$

Step 7. Draw the geometric primitives, which include two quadrangles $p_{1a}p_{1b}p_{1c}p_{1d}$, $p_{2a}p_{2b}p_{2c}p_{2d}$ and three triangles $p_{1d}p_{2a}p$, $p_{1c}p_{2b}p$, $p_{3a}p_{3b}p_3$ using the OpenGL API.

In Step 4, in order to prevent the arrow from exceeding the second line segment, we set the length of the arrow to be the length of the second line segment of the display path if the arrow length is larger than the line segment.

6.4 Algorithm of Indoor Navigation by Augmented Reality

In this section, we summarize the processes described in the previous sections as a total process — the process of indoor navigation by augmented reality, as described in Algorithm 6.4 below.

Algorithm 6.4 Indoor navigation by augmented reality.

Input: A scene image.

Output: An augmented image.

Steps

Step 1. Obtain the user's orientation, and the user's location from the server-side system.

Step 2. Obtain the pitch angle from the orientation sensor of the client device.

Step 3. Create the projection matrix by the method described in Algorithm 6.1.

Step 4. Obtain visiting target information from the server-side system.

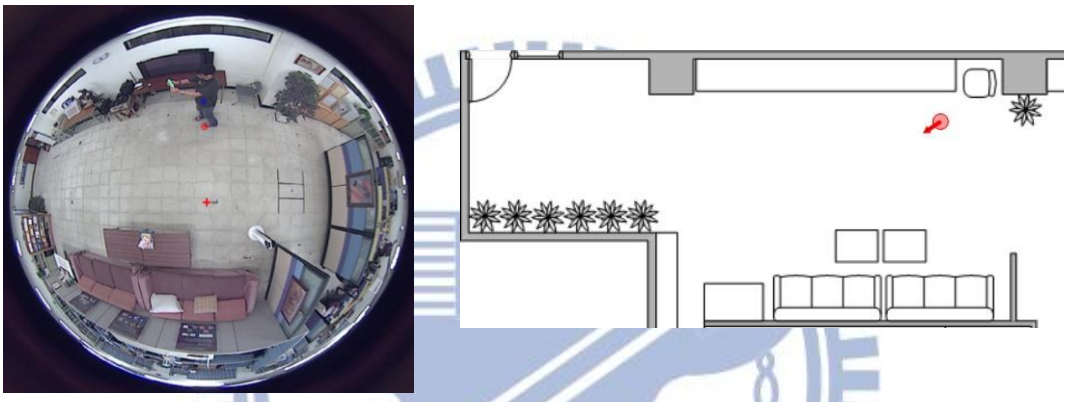
Step 5. Display all visiting targets by Algorithm 6.2.

Step 6. Search the desired destination by a keyword to obtain a planned path.

Step 7. Display the planned path by Algorithm 6.3.

6.5 Experimental Results

Figure 6.11 and Figure 6.12 show two results of overlaying visiting target information on scene images. The figure includes an omni-image captured from a fisheye camera, the detected location and orientation, and the augmented image shown on the user's mobile device. A user can understand the surrounding environment by the visiting target information on the augmented image.

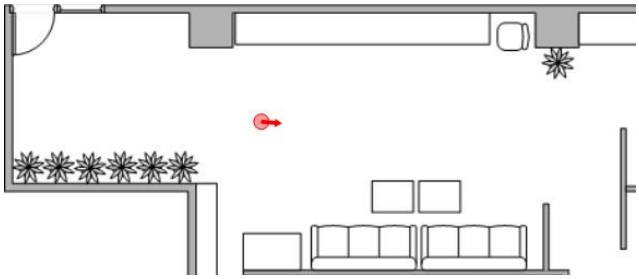


(a) (b)

(c)

Figure 6.11 An augmented image with visiting target information. (a) An omni-image. (b) Detected location and orientation. (c) The augmented image shown on user's mobile device.

(a)

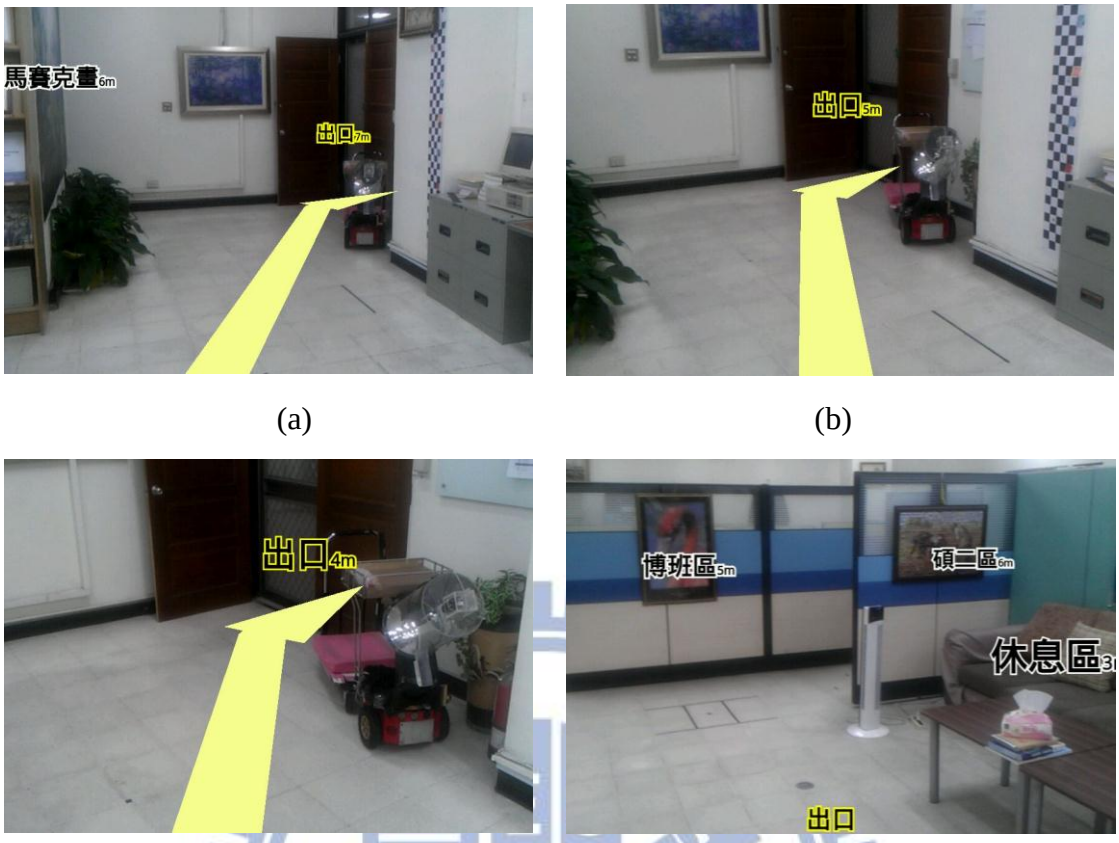


(b)

(c)

Figure 6.12 An augmented image with visiting target information. (a) An omni-image. (b) Detected location and orientation. (c) The augmented image shown on user's mobile device.

Figure 6.13 shows a result of overlaying a navigation path on scene images. The figure includes four augmented images which are at different locations and in different orientations. A user can understand how to reach the desired destination by following the navigation path shown by the arrow and the line segment. When the destination is out of the screen, the navigation path may be invisible in an augmented image. At that time, the system will display the destination on the edge of the screen to indicate the correct direction.



(a) (b)

(c) (d)

Figure 6.13 An augmented image with a navigation path. (a)(b)(c) The augmented images at three different locations. (d) When the destination is outside of the screen, the name of the destination will display on the edge of the screen (shown as the yellow stroke text); this image shows that the destination is on the rear of the user.

Chapter 7
Experimental Results and Discussions

7.1 Experimental Results

In this section, we will show some experimental results of the proposed indoor AR navigation system. The experimental environment is in the Computer Vision Lab at National Chiao Tung University. The environment map is shown in Figure 7.1, which includes eight visiting targets (shown as green regions) and two fisheye cameras (shown as blue circles).

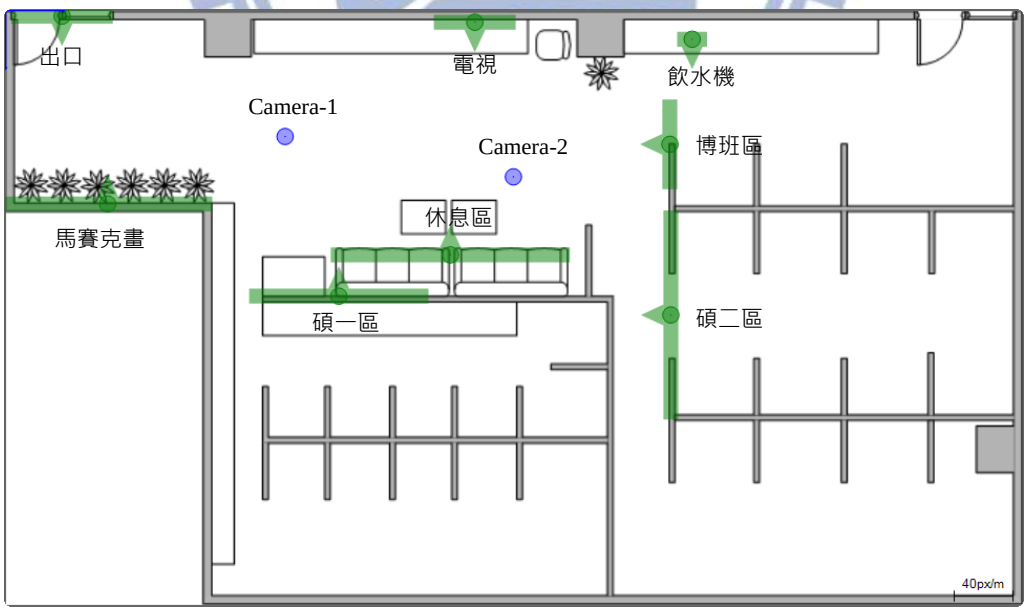


Figure 7.1 The environment map of the experimental environment.

7.1.1 Result of Real Navigations

A. Browsing visiting targets at a certain location

Figure 7.2 shows a result of browsing surrounding visiting targets at a certain location. The omni-images captured from the fisheye cameras are shown on the left-hand side of this figure, and the augmented images shown on the user's mobile device are shown on the right-hand side. At first, the user faced the left side of the experimental environment, where we can see two visiting targets displayed on the screen (as shown in Figure 7.2(a)). Then, the user began to turn to the left-hand side, and we can see that the overlaying texts are moving to the right-hand side as the user was turning (as shown in Figure 7.2(b)-(j)).

(a)

(b)

Figure 7.2 A result of browsing visiting targets at a certain location. The left-hand side is the images captured from the fisheye cameras, and the right-hand side is the augmented images shown on the user's mobile device.

(c)

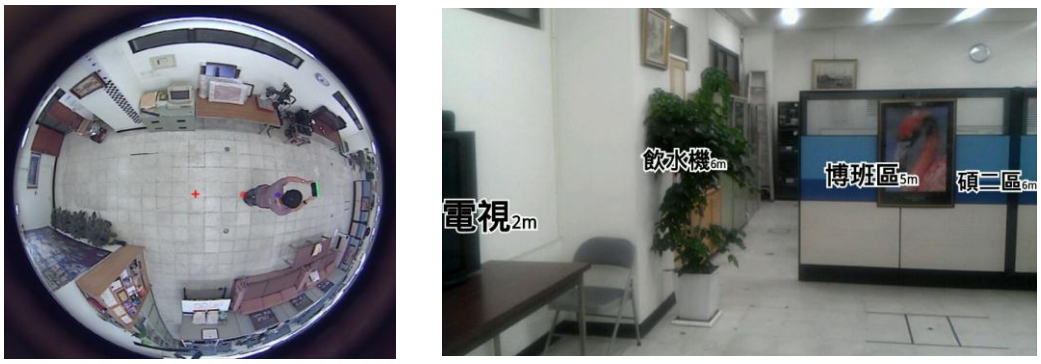
(d)

(e)

(f)

Figure 7.2 A result of browsing visiting targets at a certain location. The left-hand side is the images captured from the fisheye cameras, and the right-hand side is the augmented images shown on the user's mobile device (cont'd).

(g)



(h)

(i)

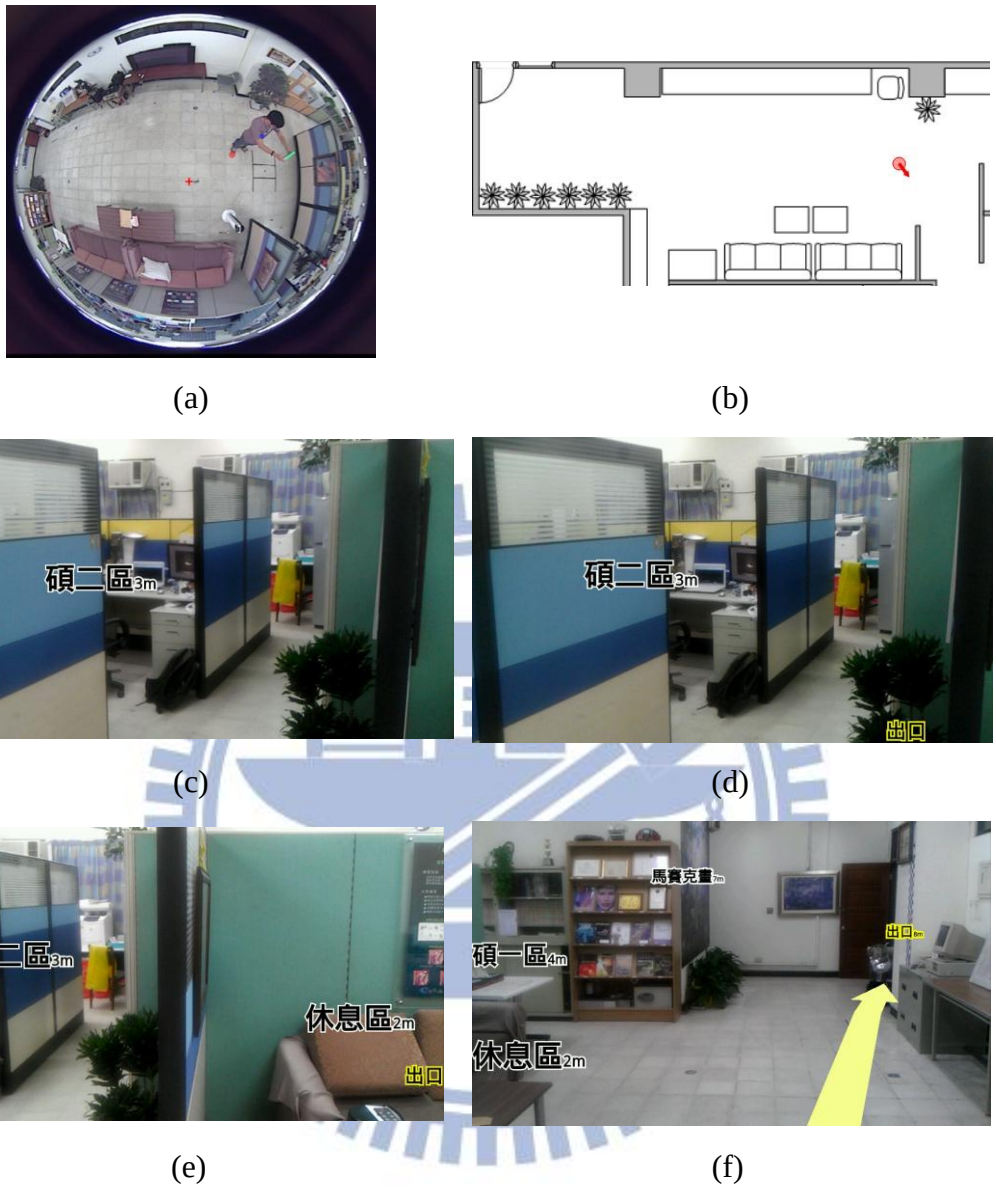
(j)

Figure 7.2 A result of browsing visiting targets at a certain location. The left-hand side is the images captured from the fisheye cameras, and the right-hand side is the augmented images shown on the user's mobile device (cont'd).

B. Navigation by a navigation path.

In this section, we show a result of navigation according to a navigation path. A user stood at a location as shown in Figure 7.3(a), and the detected location and orientation are shown in Figure 7.3(b). Figure 7.3(c) shows the augmented image seen by the user. Then, the user searched the environment map for a visiting target, and there appeared a yellow stroke text on the right-hand side of the bottom edge of the augmented image (as shown in Figure 7.3(d)). The user could then understand that the destination is on the right rear, so the user began to turn to the right-hand side. As the user was turning, we can see that the destination was moving to the right-hand side of the user (as shown in Figure 7.3(e)). Finally, the user saw the destination and the navigation path when he turned to the correct direction (as shown in Figure 7.3(f)).

Therefore, the user began to follow the navigation path to move. As shown in Figure 7.4, the user faced the left-hand side of the environment to move. When the user moved to the location as shown in Figure 7.4(b), he was closer to another camera of the environment. Therefore, the system shifted to use the other camera to track the user as shown in Figures 7.4(c) and 7.4(d). Figure 7.5 shows the four augmented images corresponding to the four locations as shown in Figures 7.4(a) through 7.4(d), respectively.



(a) (b)

(c) (d)

(e) (f)

Figure 7.3 A result of navigation by a navigation path. (a) A user was at a certain location. (b) The detected location and orientation. (c) The augmented image seen by the user. (d) The augmented image shown when the user searched a visiting target, and there is a yellow stroke text shown on the right-hand side of the bottom edge of the augmented image, which indicates the direction of the destination. (e) The augmented image shown when the user is turning to the right-hand side. (f) The augmented image shown when the user is turning to the correct direction.

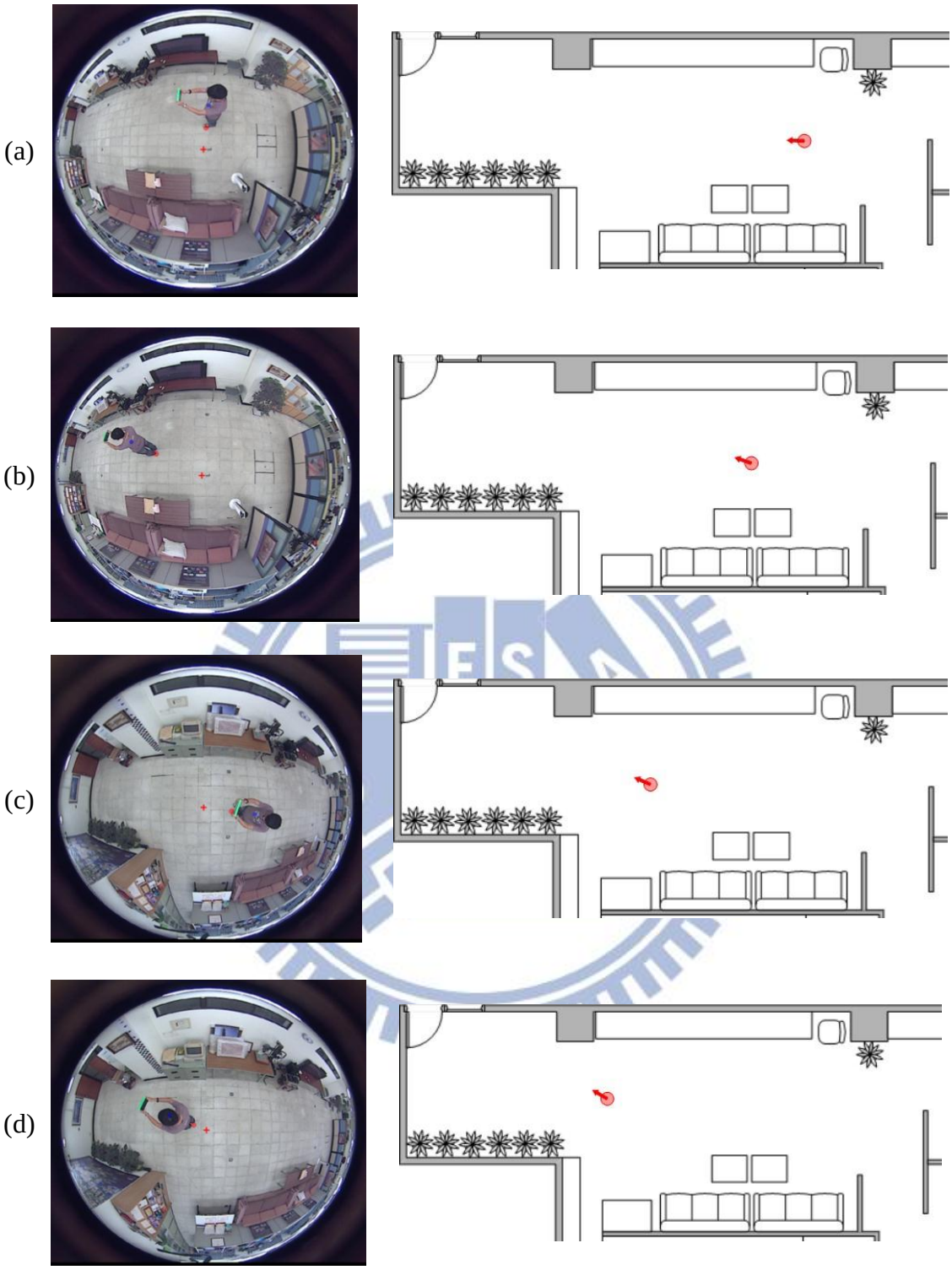


Figure 7.4 A user following the path shown in Figure 7.3(f) to move.

(a) (b)

(c) (d)

Figure 7.5 The four augmented images corresponding to the four locations as shown in Figures 7.4(a) through 7.4(d), respectively.

7.1.2 Result of Precision Measurement

We show a result of precision measurement of human location detection in this section. As shown in Figure 7.6, we chose several locations in the experimental environment, and we let a person stand at these locations and detect the locations by the proposed human location detection method. The result is shown in Table 7.1, which includes the actual locations of the chosen locations which are measured manually and the detected locations by the proposed method. The average error of the computed locations is 15cm, which is small enough for the proposed system to locate a user.

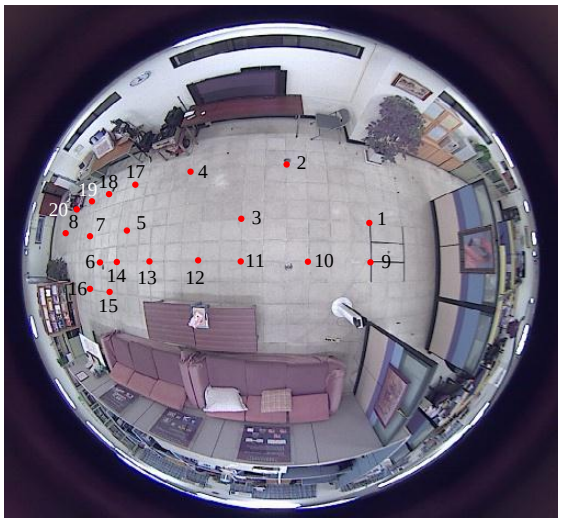


Figure 7.6 Locations used for precision measurement in the human location detection process.

Table 7.1 Error of human location detection (unit: cm)

Location #	(1) Actual		(2) Computed		Distance Error $\overline{\lvert(1)(2)\rvert}$
	x	*y*	*x*	*y*	
1	10.04	2.13	9.77	2.13	0.27
2	8.82	1.22	8.69	1.20	0.14
3	8.22	2.13	8.16	2.17	0.04
4	7.30	1.22	7.14	1.13	0.17
5	6.09	2.13	6.04	2.00	0.09
6	5.17	2.74	4.89	2.73	0.25
7	4.56	2.13	4.29	2.15	0.24
8	2.13	1.83	1.92	1.80	0.17
9	10.04	2.74	9.67	2.73	0.36
10	9.13	2.74	8.74	2.75	0.37
11	8.22	2.74	8.19	2.75	0.02
12	7.61	2.74	7.59	2.728	0.02
13	6.69	2.74	6.64	2.753	0.04
14	5.78	2.74	5.77	2.753	0.01
15	5.48	3.35	5.42	3.253	0.10
16	4.56	3.35	5.22	3.228	0.47
17	6.09	1.22	5.97	1.278	0.10
18	5.17	1.22	5.09	1.253	0.07
19	4.26	1.22	4.22	1.203	0.05
20	3.35	1.22	3.27	1.228	0.07
Average					0.15

Table 7.2 shows another result of precision measurement in the transformation from the ICS to GCS. We measured 11 line segments on the ground, and computed the length by transforming the two end points of each line from the ICS to the GCS. These lines segments we chose are shown in Figure 7.7. The average error rate is 2.79%, which is small enough and shows that the proposed transformation technique actually works for real applications.

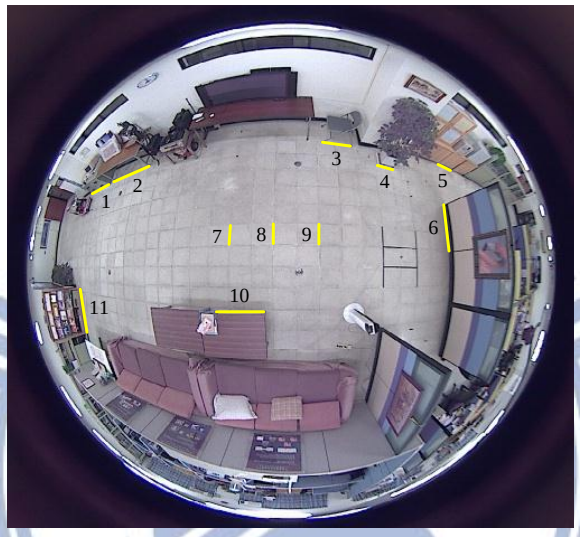


Figure 7.7 Line segments used for the line length measurement.

Table 7.2 Error of line length measurement.

Line #	(1) Actual Length (cm)	(2) Computed Length (cm)	Error (3) |(1)-(2)|	Error % $\left\|\frac{(3)}{(1)}\right\|$
1	93	96.45	3.45	3.71%
2	121	121.08	0.08	0.06%
3	48	46.50	1.50	3.13%
4	31	28.60	2.40	7.75%
5	44	41.00	3.00	6.82%
6	90	89.89	0.11	0.13%
7	30	29.56	0.44	1.47%
8	30	30.46	0.46	1.55%
9	30	30.90	0.90	3.00%
10	70	67.92	2.08	2.97%
11	121	120.87	0.13	0.11%
Average			1.32	2.79%

Table 7.3 shows the result of precision measurement for the proposed process of human orientation detection by the color edge mark. We choose six locations in the experimental environment to for this experiment. We let a person stand at these locations, and the person faced towards the mobile device to four directions at each location. Finally, we computed the orientation vector by the proposed method, and the error is the angle between the actual orientation vector and the computed orientation vector. As shown in the result, we can see that the errors are almost below 8^{o}, and the average is below 4^{o}. This shows that the proposed method actually works for ral applications. However, the errors of some cases exceed 8^{o}. The main cause of these higher errors is that the color edge mark will be projected into a small region in the omni-image when the distance between the color edge mark and the camera becomes larger, so it will not always be segmented successfully from the image completely.

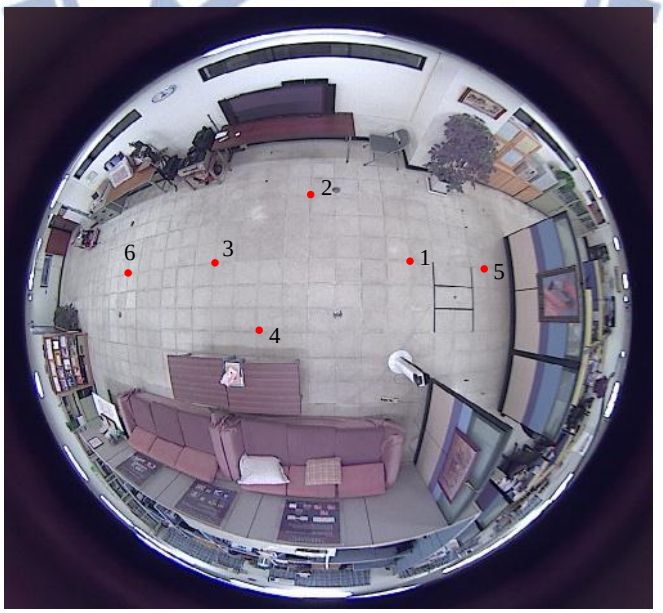


Figure 7.8 Locations used for precision measurement in human orientation detection.

Table 7.3 Error of human orientation detection.

Location #	Actual Orientation Vector		Computed Orientation Vector		Angle Error
	x	*y*	*x*	*y*	
1	-1	0	-1	-0.07	4.00°
1	0	1	0.04	1	2.29°
1	1	0	1	-0.03	1.72°
1	0	-1	-0.06	-1	3.43°
2	-1	0	-1	-0.001	0.06°
2	0	1	0.02	1	1.15°
2	1	0	1	-0.03	1.72°
2	0	-1	-0.15	-0.99	8.62°
3	-1	0	-1	-0.05	2.86°
3	0	1	-0.04	1	2.29°
3	1	0	1	-0.04	2.29°
3	0	-1	-0.03	-1	1.72°
4	-1	0	-1	-0.03	1.72°
4	0	1	-0.04	1	2.29°
4	1	0	0.99	0.1	5.77°
4	0	-1	-0.06	-1	3.43°
5	-1	0	-1	0.03	1.72°
5	0	1	0.05	1	2.86°
5	1	0	0.98	-0.2	11.53°
5	0	-1	0.02	-1	1.15°
6	-1	0	Not Detected		
6	0	1	0.06	1	3.43°
6	1	0	0.99	-0.17	9.74°
6	0	-1	-0.06	-1	3.43°
Average					3.44°

7.2 Discussions

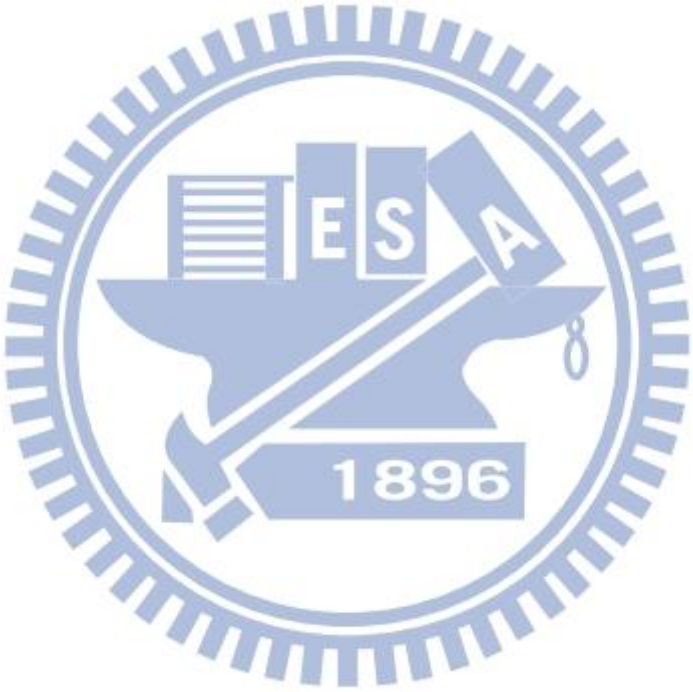
The experimental results of the proposed indoor navigation system presented previously show that we can utilize fisheye cameras to detect the user's location and orientation. Meanwhile, a user can understand the surrounding environment and conduct the navigation by the proposed AR techniques.

However, the proposed system still has some problems. As a user is moving far away from a fisheye camera, the detected locations will become more and more unstable. This is because the detected locations are computed by interpolation of four calibration points, but the pixels of farer objects will have higher distortion. Therefore, the actual distance between two neighboring pixels become larger at a far location from the camera, and the error of a few pixels might cause the interpolation result to be inaccurate. A similar problem will occur on the color edge mark detection. As the user is moving away from a fisheye camera, the region of the color edge mark in the omni-image will become smaller and hard to detect. A possible way to solve these problems is to use more cameras in the environment. Therefore, when a user moves away from a camera, it can be detected from another camera which is closer to the user. Furthermore, when a user's feet are covered by obstacles, cameras may not be able to detect correct foot locations and may result in incorrect detected locations. This problem can be solved by the same solution mentioned above, which is to use more cameras. Another possible way is to detect the head point of the user, and then we can use the height of the user to estimate the foot location.

Furthermore, our experimental environment is just a small region, so the client-side system can be connected to the server-side system through the Wi-Fi wireless network, which has a smaller access range. However, if we want to apply the proposed system in a larger environment, we might have to use a mobile

telecommunication network, such as a 3G or 4G network, to connect the both sides. The mobile telecommunication network has a larger access range than the Wi-Fi wireless network. Using mobile telecommunication networks can also reduce the costs of building Wi-Fi wireless networks.

Finally, the proposed system can handle only one user at a time. If we want to enhance the capability for multiple user usages, we have to distinguish different users in the environment. A possible solution is to analyze images captured from the mobile device, and detect the features in the images to identify different users at different locations.



Chapter 8
Conclusions and Suggestions for Future Works

8.1 Conclusions

An indoor navigation system by augmented reality and down-looking omni-vision techniques using mobile devices has been proposed. To design such a system, several techniques have been proposed as summarized in the following.

1. *A modified method for point transformation from an omni-image to the global coordinate system* has been proposed, which is modified from a space-mapping technique [16]. The proposed method can provide point transformation for larger pixel region in omni-images than the adopted method, by which we can increase the utilization of the pixels of the omni-image.
2. *A method for human localization in indoor environments* has been proposed, by which we can obtain a user's location and orientation in an indoor environment. The orientation detection algorithm integrates three different techniques to detect the orientation of a user, and each of the techniques can make up the deficiencies of the others.
3. *A method for path planning for indoor environments* has been proposed, which is based on the analysis of the floor plan drawing of an indoor environment. By this method, the system can provide a navigation path starting from a user's location to his/her desired destination.
4. *A method for indoor AR navigation by overlaying visiting target information on*

the real objects in scene images has been proposed, by which a user can understand the surrounding environment in an AR way from the overlaying visiting target information.

5. *A method for indoor AR navigation by overlaying a navigation path on the floor in scene images* has been proposed, by which a user can follow a navigation path shown on the screen and reach his/her desired destination in an AR way.

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

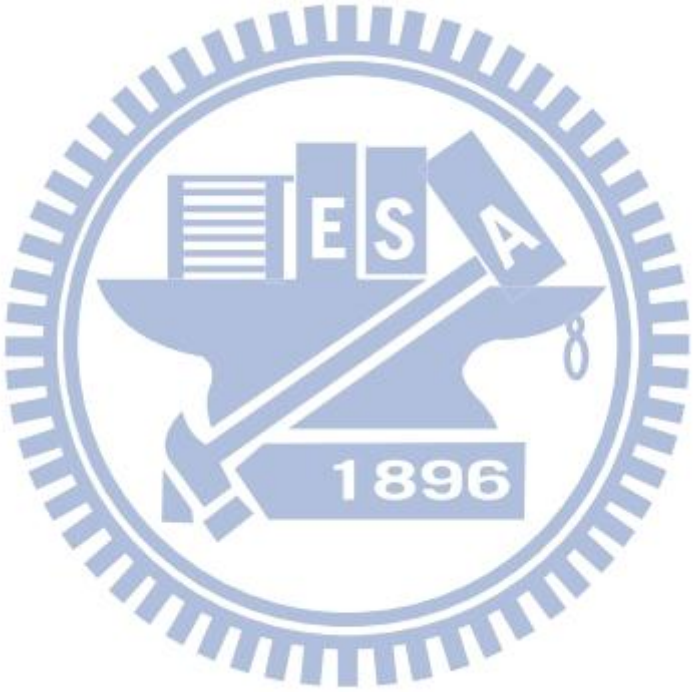
8.2 Suggestions for Future Works

According to our experience obtained in this study, several issues and possible extensions of the proposed system worth further studies are listed in the following:

1. Designing a background/foreground separation algorithm which can adapt to different lighting conditions and moving styles of non-human objects.
2. Seeking a solution to the problem of human location detection in the situation that a user's feet are covered with obstacles.
3. Proposing a method for human orientation detection with higher precision and better stability, which can be accomplished by matching the image captured from the user's mobile device with a pre-learned database to determine the orientation.
4. Providing the capability for processing multiple environment maps, which can provide human localization in different floors of an indoor environment.
5. Enhancing the system capability for indoor environments with multiple system users, which can distinguish different users in an indoor environment.
6. Including more useful information in an environment map, such as merchandise, food, etc. A user can search by a keyword for what he/she wants rather than just

searching for the name of a visiting target.

7. Collecting the images captured from cameras on users' mobile devices, and using them to establish a virtual environment database, by which users can browse an indoor environment without going there.



References

[1] C. Lukianto, C. Honniger, and H. Sternberg, "Pedestrian Smartphone-Based Indoor Navigation Using Ultra Portable Sensory Equipment," in *Proceedings of International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, Zurich, Switzerland, 2010, pp. 1-5.

[2] B. Ozdenizci, K. Ok, V. Coskun, and M. N. Aydin, "Development of an Indoor Navigation System Using NFC Technology," in *Proceedings of Fourth International Conference on Information and Computing (ICIC)*, Phuket Island, Thailand, 2011, pp. 11-14.

[3] L. C. Huey, P. Sebastian, and M. Drieberg, "Augmented Reality Based Indoor Positioning Navigation Tool," in *Proceedings of IEEE Conference on Open Systems (ICOS)*, Langkawi, Malaysia, 2011, pp. 256 - 260.

[4] A. Mulloni, D. Wagner, D. Schmalsteig, and I. Barakonyi, "Indoor Positioning and Navigation with Camera Phones," *Pervasive Computing, IEEE*, vol. 8, pp. 22-31, 2009.

[5] M. Werner, M. Kessel, and C. Marouane, "Indoor positioning using smartphone camera," in *Proceedings of International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, Guimaraes, Portugal, 2011, pp. 1-6.

[6] H. Hile and G. Borriello, "Positioning and Orientation in Indoor Environments Using Camera Phones," *IEEE Computer Graphics and Applications*, vol. 28, pp. 32-39, 2008.

[7] S. Henderson and S. Feiner, "Exploring the Benefits of Augmented Reality Documentation for Maintenance and Repair," *IEEE Transactions on*

Visualization and Computer Graphics, vol. 17, pp. 1355 - 1368, 2011.

[8] M. C. Juan, C. Botella, M. Alcaniz, R. Banos, C. Carrion, M. Melero, and J. A. Lozano, "An Augmented Reality System for treating psychological disorders: Application to phobia to cockroaches," in *Proceedings of the Third IEEE and ACM International Symposium on Mixed and Augmented Reality*, Arlington, USA, 2004, pp. 256 - 257.

[9] D. Kalkofen, E. Mendez, and D. Schmalstieg, "Interactive Focus and Context Visualization for Augmented Reality," in *Proceedings of IEEE and ACM International Symposium on Mixed and Augmented Reality*, Nara, Japan, 2007, pp. 191-201.

[10] K. Jongbae and J. Heesung, "Vision-Based Location Positioning using Augmented Reality for Indoor Navigation," *IEEE Transactions on Consumer Electronics*, vol. 54, pp. 954-962, 2008.

[11] T. Miyashita, P. Meier, T. Tachikawa, S. Orlic, T. Eble, V. Scholz, A. Gapel, O. Gerl, S. Arnaudov, and S. Lieberknecht, "An Augmented Reality Museum Guide," in *Proceedings of IEEE International Symposium on Mixed and Augmented Reality*, Cambridge, United Kingdom, 2008, pp. 103-106.

[12] H. C. Chen and W. H. Tsai, "Optimal security patrolling by multiple vision-based autonomous vehicles with omni-monitoring from the ceiling," in *Proceedings of 2008 International Computer Symposium*, Taipei, Taiwan, Republic of China, 2008, pp. 196-201.

[13] J. Borenstein and Y. Koren, "The Vector Field Histogram-Fast Obstacle Avoidance for Mobile Robots," *IEEE Transactions on Robotics and Automation*, vol. 7, pp. 278-288, 1991.

[14] J. Y. Hwang, J. S. Kim, S. S. Lim, and K. H. Park, "A Fast Path Planning by Path Graph Optimization," *IEEE Transactions on Systems, Man and Cybernetics, Part*

A: Systems and Humans, vol. 33, pp. 121-129, 2003.

[15] J. Bruce and M. Veloso, "Real-Time Randomized Path Planning for Robot Navigation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Lausanne, Switzerland, 2002, pp. 2383 - 2388.

[16] T. Akenine-Moller, E. Haines, and N. Hoffman, "Pespective Projection," in *Real-Time Rendering*, Third Edition, T. Akenine-Moller, ed., 2008, pp. 92-97.

[17] A. Senior, A. Hampapur, Y.-l. Tian, L. Brown, S. Pankanti, and R. Bolle, "Appearance Models for Occlusion Handling," in *Proceedings of 2nd IEEE Workshop on Performance Evaluation of Tracking and Surveillance*, Hawaii, USA, 2001.