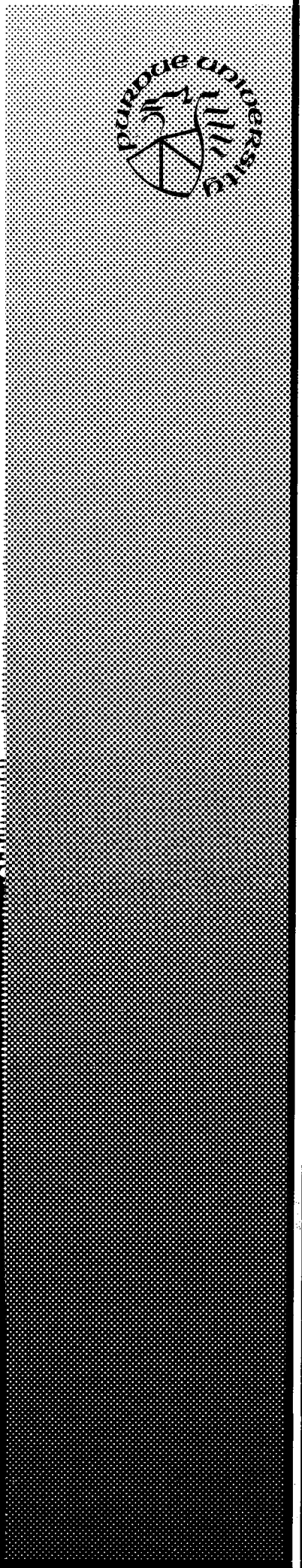
5-1-1989

Vision-Guided Mobile Robot Navigation

C. Lopez-Abadia
Purdue University

A. C. Kak
Purdue University



Vision-Guided Mobile Robot Navigation

C. López-Abadía
A. C. Kak

TR-EE 89-34
May 1989

Robot Vision Lab

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

VISION-GUIDED MOBILE ROBOT NAVIGATION

by

C. López-Abadía

and

A. C. Kak

Robot Vision Lab
School of Electrical Engineering
Purdue University
W. Lafayette, IN 47907

Technical Report TR-EE-89-34

May 1989

ACKNOWLEDGEMENT

The authors would like to express their special appreciation to Matt Carroll and Jeff Lewis, both Senior Research Engineers, for their indispensable contributions to the hardware and software aspects of the mobile robotics program at RVL. The authors would also like to thank Min Meng, currently a doctoral student exploring the use of neural networks for navigation, for her help with the production of this report.

TABLE OF CONTENTS

Page

ABSTRACT

This report discusses the use of vision feedback for autonomous navigation by a mobile robot in indoor environments. In particular, we have discussed in detail the issues of camera calibration and how binocular and monocular vision may be utilized for self-location by the robot. A noteworthy feature of monocular vision is that the camera image is compared with a CAD model of the interior of the hallways using the PSEIKI reasoning system; this reasoning system allows the comparison to take place at different levels of geometric detail.

CHAPTER 1

INTRODUCTION

Intelligent mobile robots are expected to be useful for applications ranging from factory-floor material handling and transfer to agricultural harvesting and planetary exploration. The problem of autonomous navigation in complex dynamic environments by such robots presents substantial engineering challenges in the areas of planning, perception, navigation and control.

One of the first mobile robots, named "Shakey", was the result of research conducted from 1966 through 1972 at SRI [1]. It was capable of navigating through a set of interconnected rooms in a contrived static environment and performing simple tasks such as pushing a box from one place to another. The focus of the research was on planning the robot actions by means of a hierarchy of computer programs that enabled it to perform the tasks requested. Shakey's sensory system used a black and white video camera, a range finder and several "cat-whisker" touch-sensors.

Moravec [2] built an autonomous cart that navigated among obstacles exclusively by vision, deducing its own motion from the apparent 3D shift of the features around it. To obtain three dimensional spatial information it implemented a *slider stereo* vision algorithm. After each motion step (the cart moved in one meter steps) the computer slid a single camera on a track taking nine pictures at precise intervals and then used the 36 possible image pairings to estimate the 3D location of each feature.

More recently, there has been much research on mobile robots that would be capable of operating in outdoor environments. Such robots, especially when they are expected to operate at high speeds, place great demands on computational resources for the execution of perception-driven control algorithms. One example of such a robot is the Autonomous Land Vehicle developed at FMC [3], a modified armored personnel carrier that can be operated at speeds of up to 40 Km/h. Its sensor suite includes seven color cameras, an inertial navigational system, a forward looking infrared sensor and a sonic-imaging sensor. Most of the higher level cognitive processing, such as mission and route planning, landmark recognition, obstacle detection and avoidance, etc., is carried out off board by an array of workstations and other specialized hardware.

Another autonomous vehicle for outdoor applications is the CMU NavLab [4]. The NavLab is a van equipped with a color camera, a laser range finder and four general purpose Sun-3 computers interconnected with an Ethernet. It is capable of traveling continuously at a speed of roughly 1 Km/h over a mapped network of sidewalks, while recognizing landmarks and detecting and avoiding obstacles at the same time. Other research projects at CMU study the problems associated with smaller mobile robots in indoor environments [5, 6].

Brooks [7, 8] has proposed a layered control architecture for mobile robots; the different layers constitute a hierarchy of behaviors, or levels of competence, instead of functional modules. In this model, the behavior corresponding to each layer subsumes the behavior corresponding to the layer below, hence the name *subsumption architecture* that is used for this approach to robot control. This contribution by Brooks is a part of the recent efforts at MIT [9] to scale down the size of mobile robots in an attempt to create cheaper and less complex platforms while attaining a reasonable level of intelligent behavior.

Ayache et al. [10] have presented an approach to visual navigation using a trinocular stereo vision system. The system computes the parameters of robot motion from 3D maps of the surroundings obtained by a stereo vision algorithm. The third camera facilitates the solution of the stereo correspondence problem and adds redundancy to the input information which in turn leads to more robust and accurate results.

The work presented in this report represents some of the early efforts at providing our mobile robot PETER, named for Programmable Engine for Terrain Exploration Research, with vision-based autonomous navigation capabilities. We have implemented both binocular and monocular vision systems suitable for hallway navigation; the former implementation is described in Chapter 5 and the latter in Chapter 6. Regardless of what kind of a vision system is used, the cameras need to be properly calibrated if the measurements performed in an image are to be translated into three dimensional measurements for the purpose of determining the location of the robot. Chapter 3 presents our work on camera calibration. In Chapter 2, the current configuration of PETER and the software developed to interface with it are described. A software architecture that can be used in conjunction with the stereo vision system is also presented.

List of References

[1] N.J. Nilsson, "Shakey the Robot", Tech. Report 323, SRI AI Center, April 1984.

[2] H.P. Moravec, *Robot Rover Visual Navigation*, UMI Research Press, 1981.

[3] L.S. McTamaney, "Real-Time Intelligent Control", *IEE Expert*, Winter 1987.

[4] Y. Goto and A. Stentz, "Mobile Robot Navigation: The CMU System", *IIEE Expert*, Winter 1987.

[5] H. P. Moravec, "The Stanford Cart and the CMU Rover", *Proceedings IEEE*, pp. 872-884, July 1983.

[6] A.E. Elfes, "A Sonar-Based Mapping and Navigation System", *IEEE Journal of Robotics and Automation*, RA-3(3), pp. 249-266, 1987.

[7] R.A. Brooks, "A Robust Layered Control System for a Mobile Robot", *IEEE Journal of Robotics and Automation*, RA-2, pp. 14-23, April 1986.

[8] R.A. Brooks, J.H. Connell and A. M. Flynn, "A Mobile Robot with Onboard Parallel Processor and a Large Workspace Arm", *Proceedings AAAI Conference*, pp. 1096-1100, 1986.

[9] A. M. Flynn and R.A. Brooks, "MIT Mobile Robots - What's Next?", *Proceedings IROS, TokioP, pp. 611-617, 1988.*

[10] N. Ayache, O.D. Faugeras, F. Lustman and Z. Zhang, "Visual Navigation of a Mobile Robot: Recent Steps", *Proceedings IROS, Tokio*, pp. 651-658, 1988.

CHAPTER 2

HARDWARE AND SYSTEM ORGANIZATION

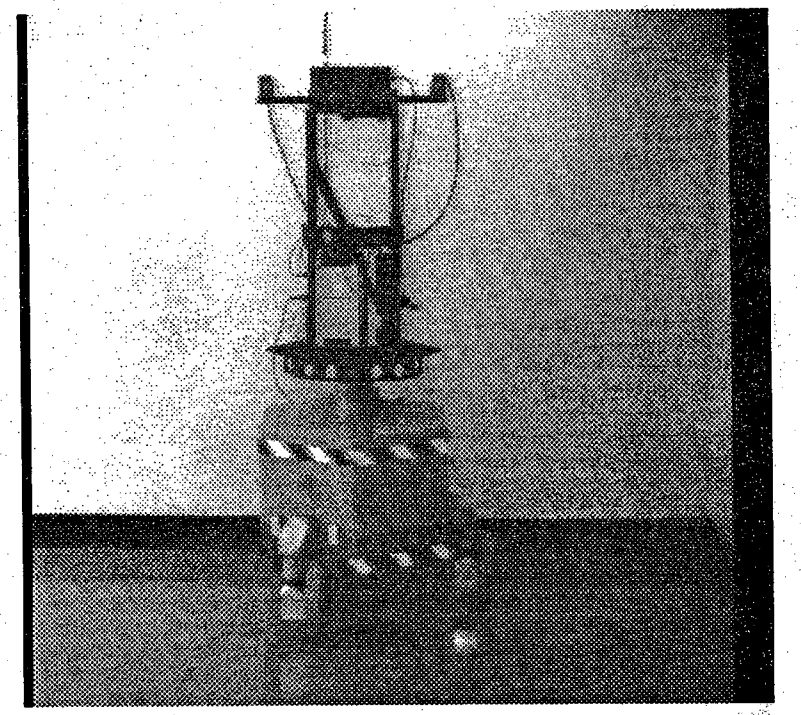
In this chapter, the various components of PETER and their interfaces with the rest of the system are described. An overview of the overall system architecture, that includes both PETER and off-board processors, is also presented.

2.1 HARDWARE CONFIGURATION

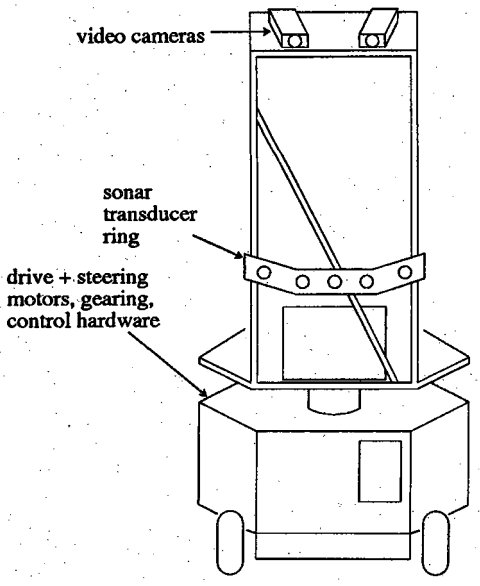
The mobile robot PETER is built on a Cybermation K2A platform. The platform has a three-wheel drive system in which all the wheels are locked together for both steering and driving. Thus, when the robot executes a turn, all three wheels turn in unison and trace parallel paths with respect to one another. The result of this geometry is that the platform itself does not rotate as the turn is executed. A turret was built on top of the platform for mounting the sensors and the on-board processing hardware. Figure 2.1 shows a picture of PETER.

At this time, all vision-related processing is carried out off-board on a SUN3 computer.* This off-board processor will be referred to as the remote host. As illustrated in Fig. 2.2 there are two communication channels between the remote host and PETER. The data link, which connects the remote host with an onboard MC68000 based computer, is used for controlling all the hardware on board. This link is an RF, full duplex, asynchronous, serial link that allows commands to be sent to PETER and status reports to be sent back to the remote host. The other link is the video channel, used for transmitting camera images from the robot to the remote host. A video switch is used under computer control to select one of the cameras (there are two mounted on the robot) for RF transmission. The output of the selected camera is broadcast by an on-board video transmitter and received by a video monitor, whose output is in turn fed to the image

*We are currently building a more powerful VME bus based processing system which would allow image digitization and much of vision processing to be carried out on-board. This system is expected to be ready in early 1990.



(a)



(b)

Figure 2.1: A picture of PETER.

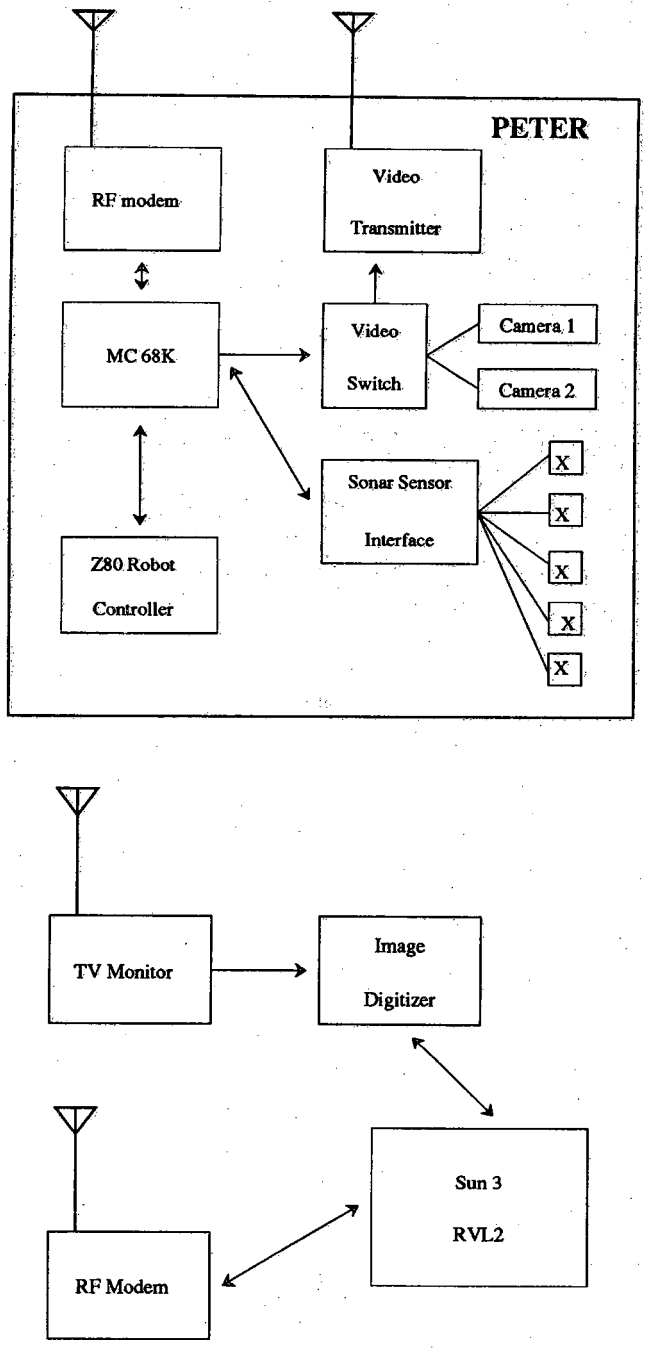


Figure 2.2: A block diagram of the equipment onboard PETER and the connections with the remote host.

digitizing hardware controlled by the remote host. There are two data links on board PETER (Fig. 2.3); these are referred to as the supervisory and control links, respectively. The supervisory link is an RS-232 asynchronous serial link that serves as a communication channel between the MC68000 and the Z80 based robot controller. The second serial on-board link is internal to the Cybermation platform and serves as a communication channel for the computers and hardware inside the platform.

The on-board MC68000 based computer controls the video switch, the sonar sensor interface and is the master of the supervisory link that communicates with the Z80 based robot controller. Upon request from the remote host, the MC68000 can select the camera output to be transmitted, take sonar readings or send any motion command to the vehicle platform. Therefore, from the standpoint of the remote host, the MC68000 processor serves as an interface with all the hardware internal to PETER.

2.2 K2A PLATFORM INTERFACE

As mentioned in the preceding section, the vehicle platform is controlled by a Z80 computer. The only interface between the robot controller and the outside world is the supervisory link, with the MC68000 processor on-board PETER as its master. The communication protocol on both the supervisory and control links is master/slave in nature, and has only two message formats: a request for data from a slave computer and a transmission of data to a slave. The transmission of data to a slave is in ASCII hex and it is structured as shown in Fig. 2.4a. For example:

:020100030102F7<CR><LF>

would transmit two bytes, 01 into address 0100 Hex and 02 into the address 0101 Hex of the slave computer 03 Hex. If slave number 3 received this message properly and calculated the correct checksum (F7H), after the <CR> was received, it would place the two bytes in memory and transmit the check sum back to the master as a single 8 bit byte (not in ASCII hex).

The message format for a request for data from a slave computer is shown in Fig. 2.4b. During the reception of such a message the slave will calculate the checksum for the message as it is received. After the slave receives the message, it will immediately transmit the requested data in raw 8 bit binary bytes, subtracting each from the checksum of the received request. After the last bit of the requested data has been transmitted, the slave will append the combined checksum in the form of an 8 bit binary byte. Note that all transmissions are initiated by the master and that the messages from the master are in the form of ASCII coded Hex, as opposed to the 8 bit binary bytes form of the transmissions from a slave to the master.

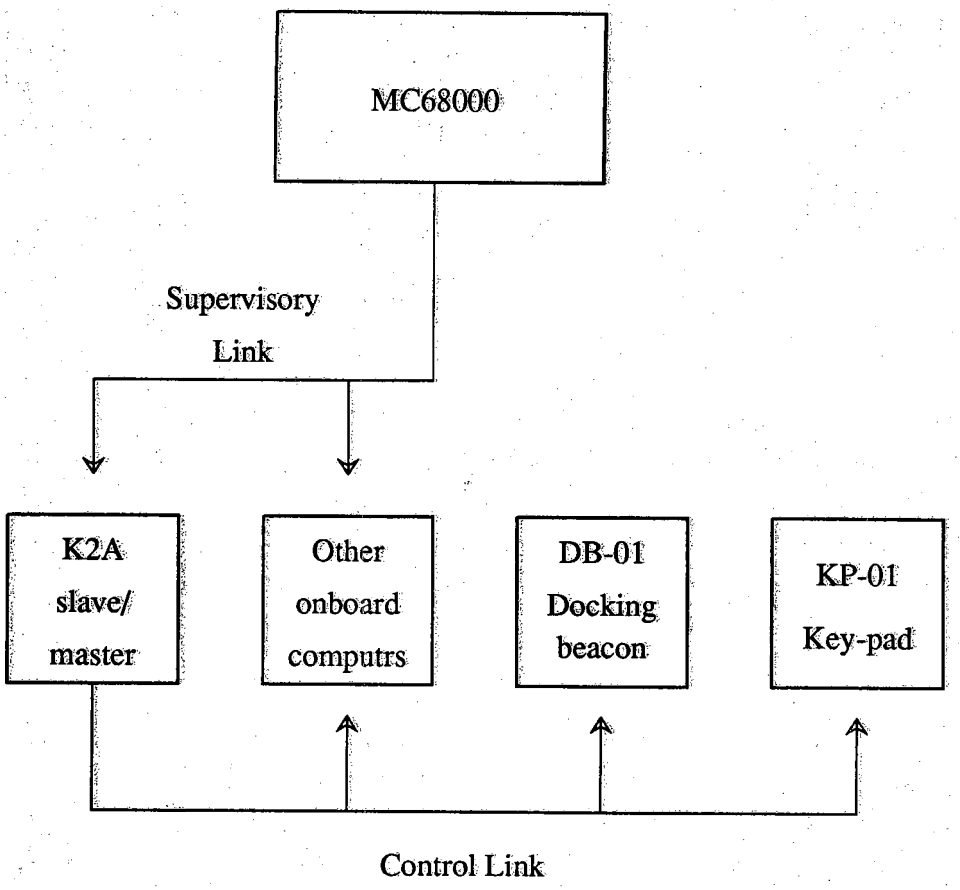


Figure 2.3: Supervisory and control links.

:NNAAAACCDD....DDSS<CR><LF> where:

:	Indicates a data transmission.
NN	Number of data bytes (2 hex digits).
AAAA	Beginning destination address (4 hex digits).
CC	Slave computer number (2 hex digits).
DD....DD	Data (2 hex digits / byte).
SS	Check sum for all digit pairs, calculated by subtraction starting with zero.
<CR>	Terminator (required).
<LF>	Optional for ease of monitoring.

(a)

;NNAAAACC<CR><LF> where:

;	Indicates a data request.
NN	Number of data bytes requested (2 hex digits).
AAAA	Beginning address of data (4 hex digits).
CC	Slave computer number (2 hex digits).
<CR>	Separator.
<LF>	Display aid for monitoring.

(b)

Figure 2.4: K2A message formats.

Using these two message protocols, the master of the supervisory link (the MC68000 processor) can read and write data in a 64K area of the slave Z80 robot controller. The actual effect of data transmission will be determined by the software running in the slave. There are two general types of data: control and parametric. Control data determines what algorithms the controller will execute. Parametric data is then used by the operating algorithm to perform the desired function. For example, writing the MODE value of the K2A platform is a control function. The following five modes are available:

MODE	
HALT	Vehicle halted.
MANUAL	Torque commanded.
AUTOMATIC	Automatic program execution.
MAN/REV	Torque commanded, reverse.
CLOSED LOOP	Velocity commanded.

An example of parametric data is the value of the drive speed when the robot is operating in CLOSED LOOP mode.

In order to be able to command PETER to perform the desired motions, a program was developed that allows the MC68000 computer to interpret a sequence of up to 100 motion commands sent by the remote computer. The program translates the motion commands into instructions executable by the K2A and writes all necessary control and parametric data into the memory of the Z80 controller so that the motion sequence can be executed. The program first places the robot in HALT mode and then, after successfully loading all the instructions and parameters, it executes them in AUTOMATIC mode; finally the robot is left in HALT mode. Table 2.1 lists the motion commands that can be interpreted by the program.

2.3 SOFTWARE ARCHITECTURE

The system organization, in form of the components or functional modules and their interconnections, is shown in Fig. 2.5. In the remainder of this section, the function of each module will be outlined; subsequently the interconnections between the modules will be explained.

The job of the **Cartographer** consists of maintaining the *a-priori* environment map and passing the information to the modules that request it. Different map information structures may be required by different modules or even by the same module. For example, the PSEIKI system described in Chapter 6 for CAD-driven interpretation of monocular vision data, needs a 3-D solid model of the hallways for generating a synthetic image

Table 2.1: Commands understood by the K2A interface. In the conditional instructions result is the value of the last READ, WRITE, ADD or SUB operation.

Command	Description
END	End of program
RUN speed X Y	Move forward to (X,Y)
TURN azimuth	Turn to azimuth
WAIT time	Wait for time
BACK speed X Y	Move backwards to (X,Y)
WRITEB address byte	Write byte at address
WRITEW address word	Write word at address
READB address byte	Read byte from address
READW address word	Read word from address
JUMP step	Jump to program step
JUMPGT step value	Jump if result > value
JUMPLT step value	Jump if result < value
JUMPEQ step value	Jump if result = value
CALL step	Call to program step
CALLGT step value	Call if result > value
CALLLT step value	Call if result < value
CALLEQ step value	Call if result = value
RETURN	return from call
ADD address value	Add to variable
SUB address value	Subtract from variable
HALT	Halt-End of program
DOCK n	Drive into dock n
UNDOCK	Clear dock status n
SETXY X Y	Set X and Y position values
SETAZ azimuth	Set azimuth value
JOG speed distance	Drive straight
ALIGN	Turn turret to docking beacon

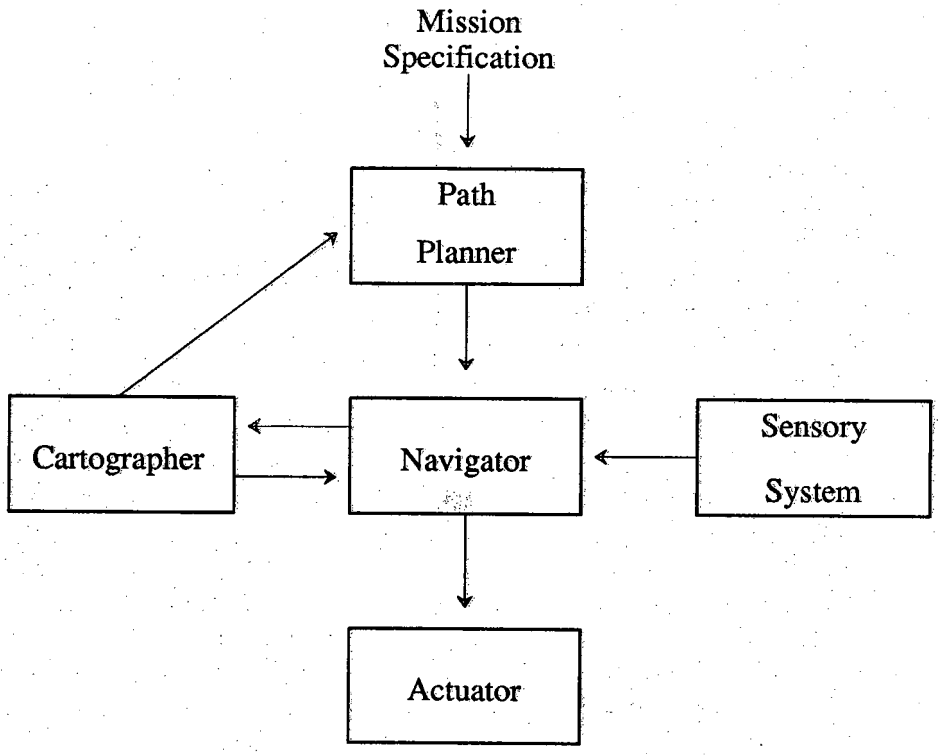


Figure 2.5: Block diagram of the system architecture.

of what the robot should see if it is where it thinks it is; the solid model of the hallways for this purpose resides in the Cartographer. On the other hand, a much simpler graph structure representing the hallway network is adequate for the Path Planner. In this graph structure, which also resides in the Cartographer, the vertices represent corners, intersections and landmarks, and the edges the straight segments of the hallways. The vertices in the hallway graph have as attributes their location coordinates and a label that allows the system to identify them by the name of the real object they represent, i.e.: Lab 180, Potter Stairs, Classroom 171, etc. The edges in the graph have as their only attribute the width of the section of hallways they represent.

The **Path Planner** receives from the user the mission objectives and defines a sequence of actions that will achieve the goals. In the context of hallway navigation, a realistic mission statement could be for example: "Go To Room 181." After obtaining the necessary information about room names and hallway topology from the Cartographer, an optimal path from the starting location of the robot to Room 181 is determined. The format of the path that the Navigator can use is a sequence of straight line segments, each described by the coordinates of its end points in a global or world coordinate frame.

The Cartographer provides to the Path Planner a graph data structure, an example of which is shown, superimposed on the outlines of the hallways outside the lab area, in Fig. 2.6. The first task of the Path Planner is to augment the given graph with two more nodes representing the present robot location and the destination location, producing an enlarged graph, such as the one shown in Fig. 2.7. In order to determine the vertices that are the closest neighbors of the added nodes in the augmented graph, a search must be carried out until a pair of adjacent vertices V_1 and V_2 is found that meet the following conditions:

- The perpendicular distance d from the location P, corresponding to the new vertex V, to the line L defined by the points P_1 and P_2, these physical points corresponding to the vertices V_1 and V_2, respectively, must be less than half the width W associated with the edge E incident at both V_1 and V_2.

- The perpendicular projection of P on line L must lie between the locations P_1 and P_2 of the end-points of the same line.

The first condition ensures that the point P is within half the hallway width of the centerline of the hallway segment represented by the edge E. The second condition establishes that P is between P_1 and P_2 along the hallway segment. In order to test for these conditions, the coordinates of P are computed in a local coordinate frame centered at P_1 with the x axis passing through P_2:

$$\begin{bmatrix} P'_x \\ P'_y \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -sin\theta & \cos\theta \end{bmatrix} \left(\begin{bmatrix} P_x \\ P_y \end{bmatrix} - \begin{bmatrix} P_{1x} \\ P_{1y} \end{bmatrix} \right) \tag{2.1}$$

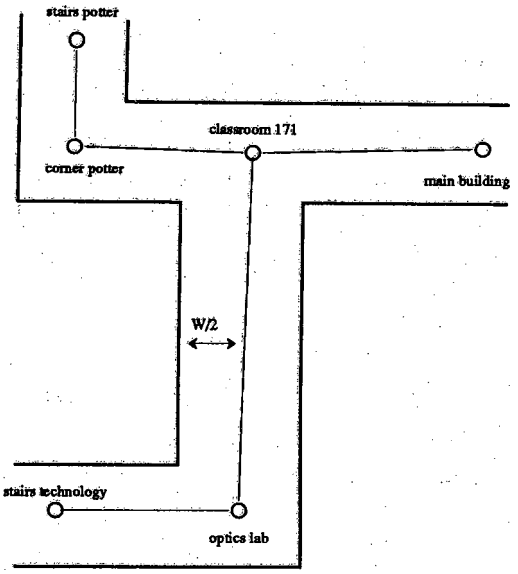


Figure 2.6: Representation of part of the EE annex as a graph.

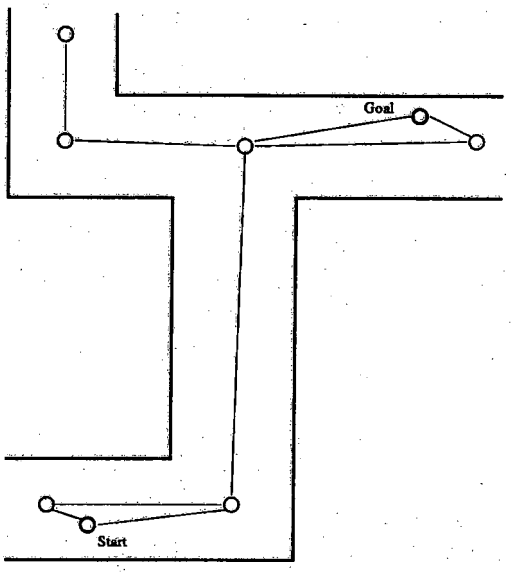


Figure 2.7: Result of adding the initial and desired locations of the robot to the hallway description.

where (P_x, P_y) and (P'_x, P'_y) are the coordinates of the point P in the world and local coordinate frames, respectively, and (P_{1x}, P_{1y}) are the coordinates of the point P_1 in the world frame. Furthermore,

$$\cos\theta = \frac{P_{2x} - P_{1x}}{|\mathbf{P}_2 - \mathbf{P}_1|} \qquad (2.2)$$

$$\sin\theta = \frac{P_{2y} - P_{1y}}{|\mathbf{P}_2 - \mathbf{P}_1|}$$

In the local coordinate frame, the two conditions stated above for determining the nearest neighbors of a new vertex can be expressed as (Figs. 2.8 and 2.9)

$$0 \leq P'_x \leq |\mathbf{P}_2 - \mathbf{P}_1| \qquad (2.3)$$

$$-\frac{W}{2} \leq P'_y \leq \frac{W}{2}$$

Once the start and the destination nodes have been added to the hallway graph, the shortest path between them is found using Dijkstra's algorithm (see appendix A). Although not currently implemented in our path planner, some applications may require that we use other than the shortest distance criterion for optimal path selection; such alternative criteria could for example be the abundance of landmarks or the likelihood of the absence of obstacles. Inclusion of such criteria would, of course, improve the robustness of the system but at the cost of a more sophisticated path planner.

After the shortest path is determined, each linear segment of the path between the nodes of the graph is examined to make certain that it does not exceed a certain pre-set threshold; if it does, new nodes are inserted so that the condition on maximum allowable length of a path between any two neighboring nodes is satisfied. The basic reason for inserting new nodes along straight segments is that it is at the locations corresponding to the nodes that the robot is required to verify its position by using vision feedback. If the length of a straight segment between the neighboring nodes is too great, excessive drift in robot location due to odometry errors may jeopardize the ability of the system to use vision for accurate self-location. The maximum allowable length of a straight line segment is a function of the vision algorithm used and the extent of the deviations from the straight line path for collision avoidance. The maximum such length for stereo-vision based self-location is 4 m at this time. When the PSEIKI system is used, this length threshold is increased to 6 m. An example of the output of the Path Planner is shown in Fig. 2.10, where again the graph data structure output by the planner is superimposed on an outline of the hallways.

The **Sensory System** must control the on-board sensors and analyze their output. At this point the principal sensory input of PETER is vision. The use of the ultrasound range sensors is primarily limited to near obstacle avoidance.

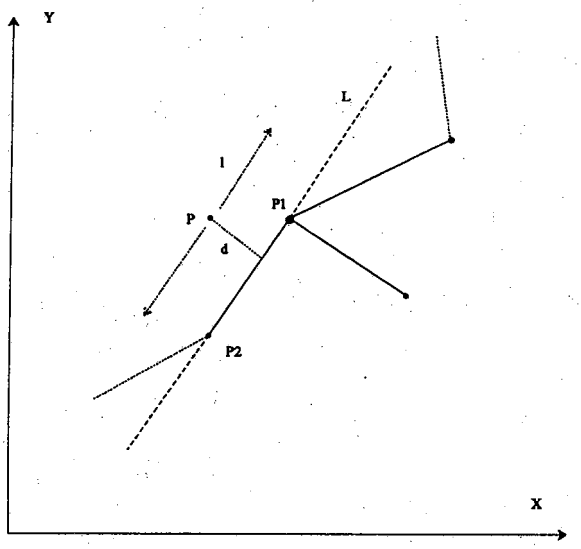


Figure 2.8: To determine in what segment of the hallway network is a point P the perpendicular and parallel distances, d and l respectively, with respect to each line L have to be computed.

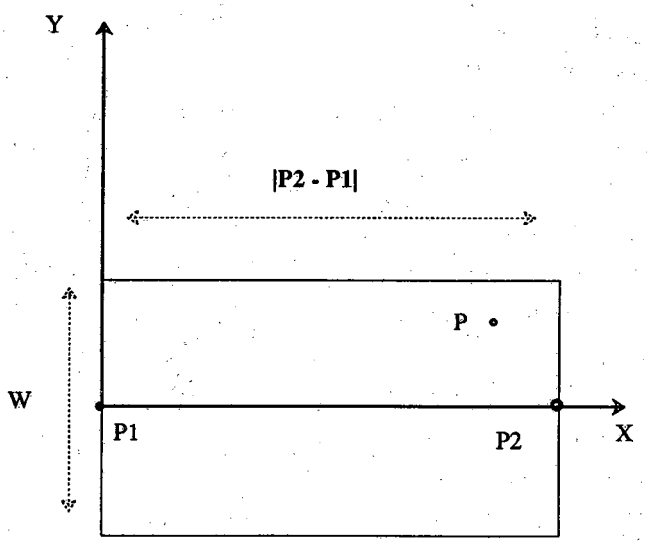
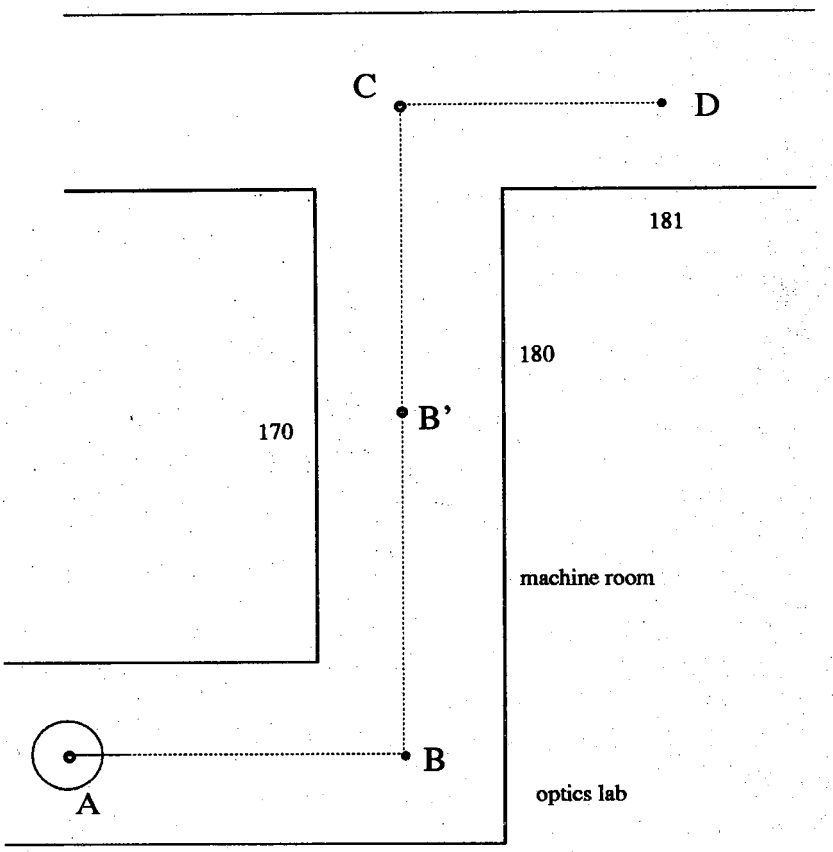


Figure 2.9: Graphical representation of the tests on a point P once it is expressed in a more convenient coordinate system.



Path: A, B, B′, C, D

Figure 2.10: Output of the Path Planner for the request "go to room 175" when the starting location is A. Note that the segment BC was broken into the two subsegments BB′ and B′C to prevent excessive drift in the robot location during its traversal.

We have implemented two approaches to the use of the visual information available to PETER. The one described in Chapters 4 and 5 uses the information of both on-board cameras to provide a 3D description of the hallway scene using a passive stereo algorithm. The scene description provided by the stereo vision system is compared by the Navigator with map information from the Cartographer to determine the current location of the robot. Note that with stereo vision, the correspondence between the scene and map features has to be determined in the 3D world domain. The other approach, described in Chapter 6, uses the PSEIKI system [1,2] to interpret the image of a single camera for solving the self-location problem. In PSEIKI, an expected map of what the robot should see is generated by rendering a CAD model of the hallways; this expected image is then compared with the actual camera image and inferences drawn about the actual current location of the robot.

The **Actuator** executes the required motion primitives received from the Navigator. The Actuator understands a number of primitive motion commands such as (turn θ°) and (move @speed to X, Y). The complete list of motion commands that the Actuator can execute is shown in Table 2.1.

The **Navigator** orchestrates all tasks necessary to execute a given plan. Upon the receipt of a path from the Path Planner, the Navigator can start executing that path by sending motion commands to the Actuator and monitoring the progress by querying the Sensory System and the Cartographer. Note that if the odometry were the only source of information about the location of the robot, the uncertainty in its position and orientation would increase steadily as the robot moved away from its initial location. Without the aid of any sensory feedback this uncertainty would grow to the point where the robot would be effectively lost -- lost in the sense that any subsequent sensor-based processing might not allow the robot to figure out its true location. As was mentioned before, it is for this reason that the planned path is divided into short enough segments so that the uncertainty at the end of each segment would not be too large to preclude a vision-based exercise in self-location and the zeroing out of whatever uncertainty might be associated with the robot at the time it used vision to locate itself. Vision-based updating of the location of the robot takes place by comparing the vision data supplied by the Sensory System with a map of the environment supplied by the Cartographer (Fig. 2.11). After such an update, the Navigator commands the Actuator module to the location corresponding to the next node in the planned path.

Although not implemented at this time, it is hoped that in the future another important job of the Navigator would be to help the Cartographer update its map of the environment if the Sensory System reports the presence of objects and/or features not in the map supplied originally by the Cartographer. Of course, implementation of this idea is going to raise important questions regarding how to distinguish between transient objects/features in the environment -- such as humans -- especially when they appear to

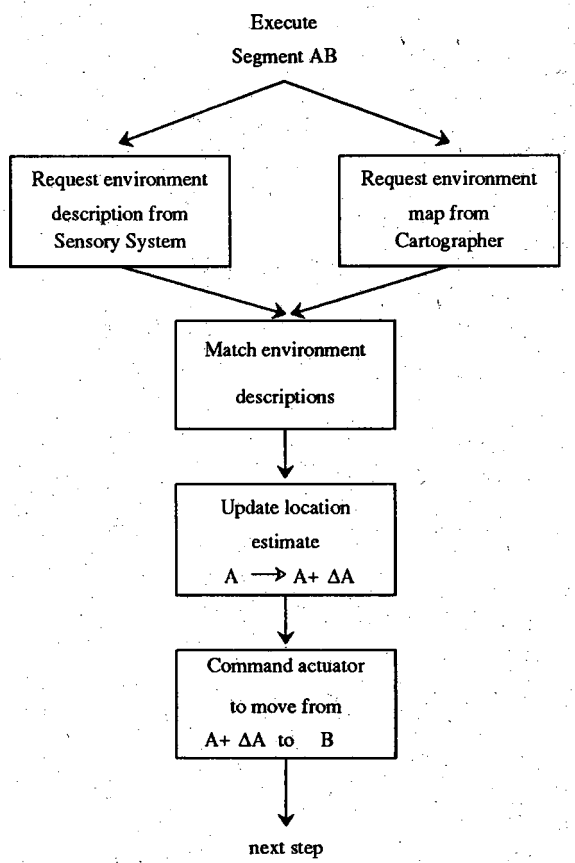


Figure 2.11: Sequence of actions performed by the Navigator in order to execute a segment of the path.

be stationary and those objects/features that are truly permanent.

REFERENCES

[1] K.M. Andress and A.C. Kak, "Evidence accumulation & Flow of Control in a Hierarchical Spatial Reasoning System", *AI Magazine*, pp. 75-94, Summer 1988.

[2] K.M. Andress and A.C. Kak, "The PSEIKI Report -- Version 2", Purdue University, School of Electrical Engineering, TR-EE 88-9, 1988.

[3] J.A. Bondy and U.S.R. Murty, *Graph Theory with Applications*, North-Holland, 1976.

CHAPTER 3

CAMERA CALIBRATION

Camera calibration is a fundamental pre-requisite to the use of vision, be it in stereoptic mode or in monocular mode. After camera calibration, for a given pixel in the image plane we can compute the line-of-sight to the scene point to which that pixel corresponds. The calibration parameters associated with a pair of cameras tell us how to translate the disparity field generated by fusing the camera images into a distance map of the world.

Usually, calibration is carried out by assuming a model for the process of image formation by the cameras. A number of models are available for this purpose, the two best known being the pinhole model, which is much used for fixed focal-length cameras with high quality lenses, and the two-plane model, used when we cannot assume the passage of all the rays through a single point. We will show that the pinhole model is appropriate for our purposes, in the sense that the errors in depth values using this model are a small enough fraction of the range values we are interested in.

3.1 CALIBRATION METHODS: AN OVERVIEW

The process of image formation can be seen as a projection of a three dimensional space into a plane, which is only two dimensional. By assuming a suitable model for this process and computing the parameters of the model via calibration, it is possible to at least determine the line-of-sight to a point in the 3D space from its image coordinates. Given such lines of sight from more than one vantage point, it is even possible to compute the location of a scene point. Modeling captures the physics of image formation and a calibration procedure then tells us how to compute the various parameters of the model. Evidently, the model used must be applicable to the cameras and lenses used in a given system. Of course, for any given model, it is possible to devise various calibration strategies of varying degrees of accuracy.

3.1.1 Pinhole Models

This is an appropriate model to use for fixed focal length cameras with high quality lenses. In this model, the image of a scene point is located at the intersection of the image plane with a line from the scene point which passes through a point on the optic axis located one focal length away from the image plane, as shown in Fig. 3.1. In other words, all the lines of sight that form the image pass through a common point on the optic axis; this point is called either the camera lens center or the center of projection. Usually, the pinhole model is only applicable when the thin lens approximation holds. With thick lenses and when distortions are present, the pinhole model breaks down and must be supplemented with corrections terms, as, for example, has been done by Tsai [8], [9]. The pinhole model is well documented in the texts [1], [2], [3].

Yakimovsky and Cunningham [4] describe their calibration procedure using the pinhole model of a pair of cameras on board the JPL Robotics Research Vehicle. They used cameras with narrow fields of view and their lenses were highly linear, meaning that their lenses did not require radial and/or tangential correction terms.

Moravec [5] also used a pinhole model in the calibration of the CMU rover's camera. He corrected the lens distortion in his system by using two two-dimensional polynomials; one of the polynomials was used to relate the positions of the points in the image to their corresponding locations for an ideal pinhole camera of unit focal-length -- this polynomial could then be used to compute lines of sight to scene points from their image coordinates. The other polynomial represented the reverse transformation, from the ideal pinhole case to the actual image coordinates. The latter polynomial could be used for synthesizing the images of expected scenes.

Tsai [8, 9] proposed a two stage calibration procedure for computing the parameters of a pinhole model augmented with radial distortion terms. In the first stage he solves for a subset of the parameters using only linear equations and constraints among the parameters. Then in the second stage the pinhole projective equations are used to find an approximation for the second group of parameters. This approximation is used as an initial guess in a non-linear search that is required to compute their final values using the full model equations (pinhole plus distortion). A nice feature of this method is that it allows for co-planar scene points to be used for calibration, which is unlike in most pinhole based procedures.

Wong [7] described a formulation suitable for calibration of the pinhole model. It includes terms representing radial and tangential distortion. He suggested two possible procedures for calibration using this model. The very high requirements on accuracy in photogrammetry necessitate that both radial and tangential corrections be taken into account in augmenting a pinhole model; however, in computer vision applications such is

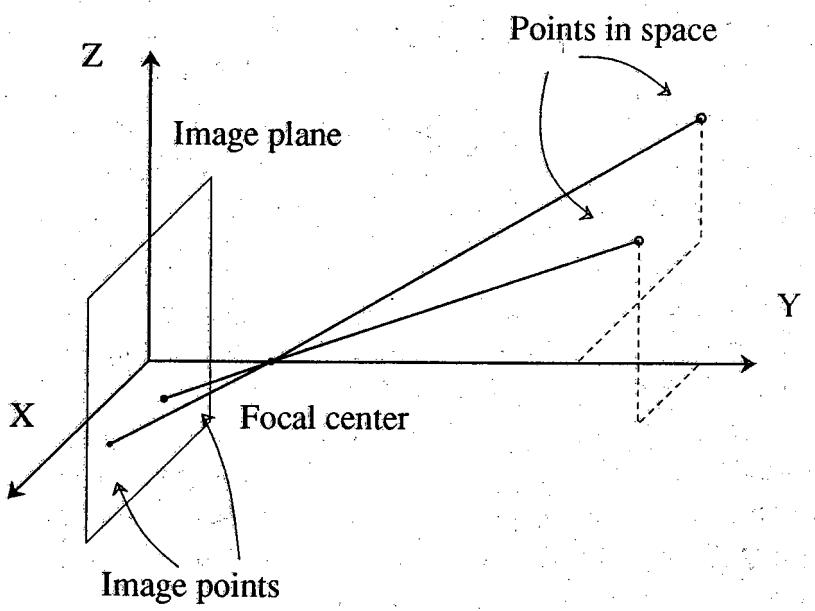


Figure 3.1: Image formation in the pinhole model.

rarely necessary and often it is sufficient to consider only the radial corrections, if needed at all.

Ganapathy [6] derived a non-iterative procedure to compute the camera parameters corresponding to a pinhole model from a given homogeneous perspective transform matrix; note that the transformation from the scene coordinates to the image coordinates in a pinhole model corresponds to a perspective transform. In the procedure advanced by Ganapathy, it is assumed that the camera has already been calibrated and the procedure is to be used to compute the physical parameters of the camera, such as the position of the camera, the orientation in terms of pan, tilt and swing angles, the location of the center of the image plane, etc.

3.1.2 Two Plane Model

In the two plane model the imaging process itself is not represented. Instead it models the *back-projection* process, or image to world transformation, as a mapping from pixels to lines of sight. This mapping is not explicitly related to any underlying physical process. Unlike in the pinhole model, in the two plane model the lines of sight are not forced to intersect at the same point. Each line of sight ray is described by its intersection with two *calibration planes* (Fig. 3.2) therefore the name of the model. The calibration procedure consists of the process of determining the parameters of the image to calibration planes mapping.

Different mapping schemes can be used. Perhaps the simplest conceptually and most accurate could be a look-up table with an entry for each pixel in the image. Each entry would indicate the corresponding line of sight. In the calibration process the intersection of these lines with the calibration planes would be measured and the entries in the table filled. This approach however could be prohibitively expensive. A more reasonable scheme is to measure the lines of sight for a number of pixels and interpolate for other points between them. The calibration process consists then of finding out the parameters of the interpolating functions.

The two-plane method is described in detail in Martins et al. [11]; they have proposed three types of interpolation. In their linear and quadratic interpolation methods, they globally fit an interpolation function to all the calibration points on each calibration plane. The functions are then used for any pixel across the entire image. The third interpolation method presented consists of tesselating each calibration plane with triangles and performing linear interpolation within each triangle. The calibration points are the vertices of each triangle. Izaguirre et al. [12] have employed the two-plane model with global polynomial interpolation in the calibration of a pair of mobile cameras. Gremban et al. [13] compared the accuracy of the method using local and global interpolation.

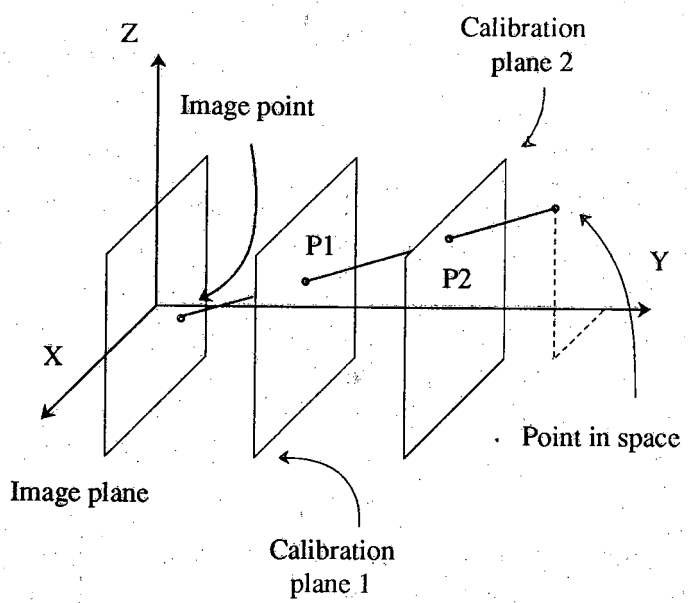


Figure 3.2: Image formation in the two plane model.

One of the advantages of the two-plane model is that it accommodates some lens distortion without the need for augmenting terms. In particular when local splines interpolation is used, the imaging system can be modeled to arbitrary accuracy by increasing the density of calibration points. On the other hand, one of its drawbacks is that it is designed to work only from image coordinates to world coordinates, making it impractical to predict the view of the world that the robot will have at a given location.

To determine the appropriateness for our application, we implemented some of the calibration methods mentioned in this section. In the next section, these methods are described in greater detail and their relative merits discussed.

3.2 CALIBRATION METHOD BASED ON IDEAL PINHOLE MODEL

In this section, a calibration procedure based on the ideal pinhole model is presented. The perspective transform equations, that is the equations that describe the imaging process using this model, are derived and a method for their solution is described.

3.2.1 Camera Model

In this subsection, we derive the perspective transform and inverse perspective transform equations in a form suitable for computer implementation. The perspective transform allows us to compute the image coordinates corresponding to an object point whose world coordinates are given. The inverse perspective maps each image point to the line of sight on which the corresponding object point must lie.

The physical location of the camera can be defined by the position of its focal center. If the world coordinate system is centered at O, let vector $\vec{C}$ define the location of the camera focal center with respect to O. We can define another coordinate system centered at $\vec{C}$ with base unit vectors $\hat{h}$, $\hat{v}$, $\hat{a}$. We will refer to this system as the camera coordinate system. Both the camera and the world coordinate systems are assumed to be orthonormal. The camera system is oriented such that $\hat{a}$ is perpendicular to the image plane and $\hat{h}$ lies along the direction of the picture scan. The image plane intersects the $\hat{a}$ axis at a distance f from $\vec{C}$ (see Fig. 3.3).

Any point on the image plane can now be referred to by its coordinates (u,v) in the image plane. The world coordinates of such an image point are given by the vector

$$\vec{C} - f\hat{a} + u\hat{h} + v\hat{v} \tag{3.1}$$

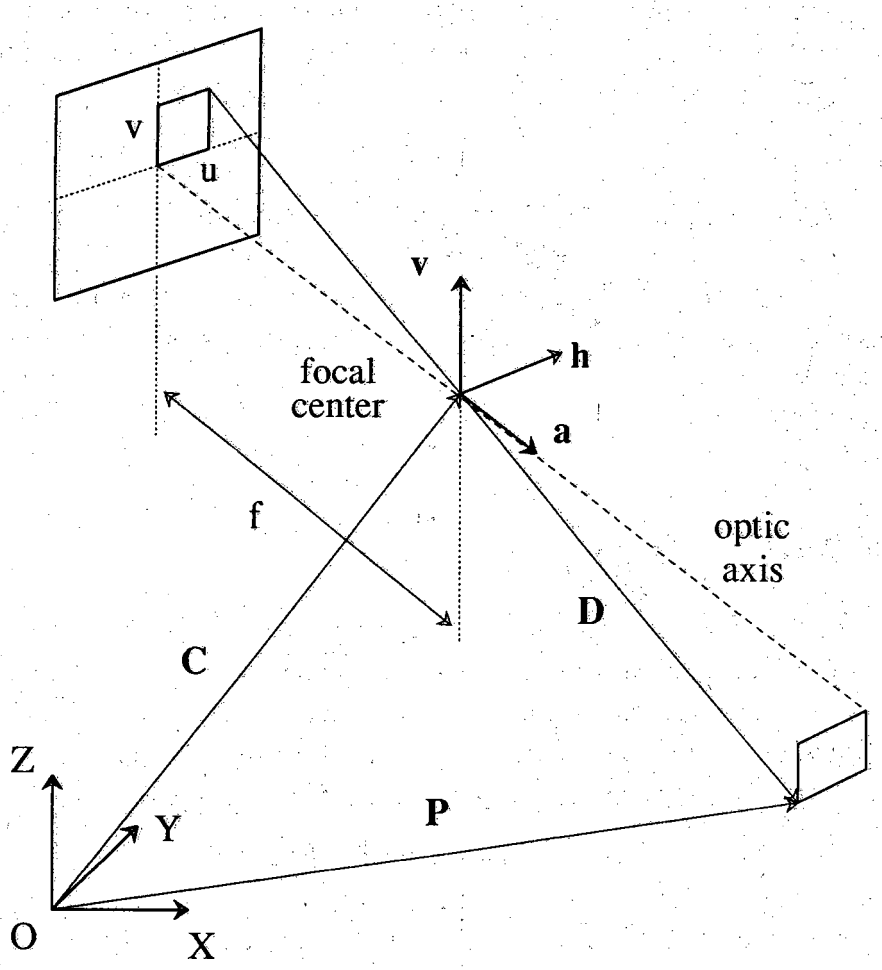


Figure 3.3: Some of the parameters of the pinhole model.

To obtain the image coordinates of a given object point $\vec{P}$ we can compare similar triangles in Fig. 3.3 and find

$$\frac{u}{f} = \frac{\vec{D}\cdot\hat{h}}{\vec{D}\cdot\hat{a}} \tag{3.2a}$$

$$\frac{v}{f} = \frac{\vec{D}\cdot\hat{v}}{\vec{D}\cdot\hat{a}} \tag{3.2b}$$

with $\vec{D} = \vec{P} - \vec{C}$. Observe that the numerators and the denominators in (3.2) are nothing but the coordinates of vector $\vec{D}$ in the $(\hat{h}, \hat{v}, \hat{a})$ frame.

In practice, the coordinates of an image point are not measured in terms of the analog entities (u, v), but by using their discrete counterparts, represented here by the indices (i, j). The analog and the discrete versions are related by

$$u = (i - i_0)\,\Delta u \tag{3.3a}$$

$$v = (j - j_0)\,\Delta v \tag{3.3b}$$

where (i_0, j_0) are the discrete coordinates of the center of the image $(u = 0, v = 0)$ and Δu and Δv stand for the sampling intervals along the $\hat{h}$ and $\hat{v}$ directions, respectively. Substituting Eq. (3.3) in Eq. (3.2) results in

$$i = \frac{f}{\Delta u}\,\frac{\vec{D}\cdot\hat{h}}{\vec{D}\cdot\hat{a}} + i_0 \tag{3.4a}$$

$$j = \frac{f}{\Delta v}\,\frac{\vec{D}\cdot\hat{v}}{\vec{D}\cdot\hat{a}} + j_0 \tag{3.4b}$$

These equations can be rewritten in a more compact form as

$$i = \frac{\vec{D}\cdot\vec{H}}{\vec{D}\cdot\hat{a}} \tag{3.5a}$$

$$j = \frac{\vec{D}\cdot\vec{V}}{\vec{D}\cdot\hat{a}} \tag{3.5b}$$

with

$$\vec{H} = \frac{f}{\Delta u}\,\hat{h} + i_0\,\hat{a} \tag{3.6a}$$

$$\vec{V} = \frac{f}{\Delta v}\,\hat{v} + j_0\,\hat{a} \tag{3.6b}$$

Although the system defined by $\vec{H}$, $\vec{V}$, $\hat{a}$ no longer represents an orthogonal coordinate frame, these vectors are still linearly independent and they form a three-dimensional basis. Therefore, the numerators and the denominators in Eq. (3.5) represent the

coordinates of the vector $\vec{D}$ in the $(\vec{H}, \vec{V}, \vec{a})$ frame, multiplied by the quantities $|\vec{H}|$, $|\vec{V}|$ and 1, respectively.

The equations in (3.5) can be recast in a more convenient matrix form with the use of homogeneous coordinates. A three dimensional point with coordinates x, y, z, is represented by its homogeneous coordinates wx, wy, wz, w where w is an arbitrary scalar. The coordinates are recovered by dividing the first three components by the last. Similarly, a two-dimensional point of coordinates u, v has wu, wv, w as homogeneous coordinates.

Using homogeneous coordinates, we can rewrite equations (3.5) as

$$\begin{bmatrix} i\,w \\ j\,w \\ w \end{bmatrix} = \begin{bmatrix} H_x & H_y & H_z & 0 \\ V_x & V_y & V_z & 0 \\ a_x & a_y & a_z & 0 \end{bmatrix} \begin{bmatrix} D_x \\ D_y \\ D_z \\ 1 \end{bmatrix} \tag{3.7}$$

Since $\vec{D} = \vec{P} - \vec{C}$ we have

$$\begin{bmatrix} D_x \\ D_y \\ D_z \end{bmatrix} = \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} - \begin{bmatrix} C_x \\ C_y \\ C_z \end{bmatrix} \tag{3.8}$$

or, in a more compact form

$$D = P - C \tag{3.9}$$

Combining (3.7) and (3.8), we can write

$$\begin{bmatrix} iw \\ jw \\ w \end{bmatrix} = \begin{bmatrix} H_x & H_y & H_z & -C'_x \\ V_x & V_y & V_z & -C'_y \\ a_x & a_y & a_z & -C'_z \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} \tag{3.10}$$

which may be expressed more compactly as

$$P_I = T\,P_W \tag{3.11}$$

where P_W is the homogeneous coordinate representation of the world point P, and P_I the homogeneous coordinate representation of the corresponding image point. The matrix relating the two is designated as T. The fourth column of T can be written in terms of C as

$$C' = R\,C \tag{3.12}$$

where R stands for the 3×3 submatrix of T:

$$R = \begin{bmatrix} H_x & H_y & H_z \\ V_x & V_y & V_z \\ a_x & a_y & a_z \end{bmatrix} \tag{3.13}$$

Matrix T is frequently referred to in the literature as camera calibration matrix or perspective transform matrix. This matrix contains all the necessary information to compute both the perspective and the inverse perspective transformations. Eq. (3.10) is used to obtain the image coordinates corresponding to an object point. To compute the line of sight corresponding to an image point, first C is calculated. From Eq. (3.12)

$$C = R^{-1} C' \tag{3.14}$$

Inverting matrix R presents no problem since it is nonsingular if the focal length f is non-zero. This can be demonstrated by noting that

$$|R| = \vec{H} \cdot (\vec{V} \times \hat{a}) \tag{3.15}$$

or from (3.6)

$$|R| = \vec{H} \cdot (\frac{f}{\Delta v} \hat{v} \times \hat{a} + j_0 \hat{a} \times \hat{a}) \tag{3.16}$$

which leads to

$$|R| = \vec{H} \cdot \frac{f}{\Delta v} \hat{h} = \frac{f^2}{\Delta u \Delta v} \tag{3.17}$$

Therefore the determinant of R is non-zero if f is non-zero.

The direction of the line-of-sight can be obtained from (3.7) by setting $w = \vec{D}\hat{a} = 1$. We can always do this since we are only interested in a direction not in the magnitude of $\vec{D}$. We have

$$P_I = \begin{bmatrix} i \\ j \\ 1 \end{bmatrix} = R D \tag{3.18}$$

and since we know that R is invertible

$$D = R^{-1} P_I \tag{3.19}$$

Note that this D does not have the correct magnitude since we artificially set $\vec{D}\hat{a} = 1$, but then we are not interested in the magnitude of D. With the D obtained from the above equation, the correct value of D, the one that would satisfy, say, Eq. (3.9), is given by μD for some value of the parameter μ. Eq. (3.9) may now be used to establish an equation for the line-of-sight to the object point corresponding to the given image point:

$$P_W = \mu D + C \quad or \quad P_W = \mu R^{-1} P_I + C \tag{3.20}$$

This represents a parametric form for the equation of the line-of-sight.

3.2.2. Calibration Procedure

The purpose of the camera calibration procedure is to obtain the values of the elements of the calibration matrix from a set of points of known image and world coordinates. In this subsection we describe a calibration procedure using the model described in the previous subsection.

The calibration matrix T is a real 3×4 matrix which represents 12 unknowns. In principle, knowledge of the image and world coordinates of 6 non-coplanar points should suffice to accomplish the calibration since each point generates two equations. In practice more than 6 points are used and a least squares optimal solution is obtained.

We should be aware that not all the twelve elements T_{mn} in T are independent. All the T_{mn} are functions of the parameters $\vec{C}$, $\hat{h}$, $\hat{v}$, $\hat{a}$, f, Δu, Δv, i_0, and j_0, and there are constraints on how these parameters appear in the expressions for T_{mn}'s. For example, f, Δu and Δv can only appear as the scaling factors $\frac{f}{\Delta u}$ and $\frac{f}{\Delta v}$ in the equations; as a consequence, the three parameters result in only two independent unknowns $k_u=\frac{f}{\Delta u}$ and $k_v=\frac{f}{\Delta v}$. It should also be noted that the nine components in the vectors $\hat{h}$, $\hat{v}$ and $\hat{a}$ are not independent, since the orthonormal coordinate system for which the vectors $\hat{h}$, $\hat{v}$ and $\hat{a}$ constitute a basis has only three degrees of freedom with regard to its orientation. The orientation of the coordinate system they define can instead be described by three angles θ, ϕ and ψ for describing the pan tilt and swing of the camera. These angles measure the consecutive rotations about the camera system axes necessary to bring it into coincidence with the world system orientation.

Therefore the 12 entries in T are combination of the following 10 independent parameters: the three components of $\vec{C}$, and θ, ϕ, ψ, k_u, k_v, i_0, j_0. We could rewrite (3.10) in terms of these independent parameters and find the optimal solution of the resulting non-linear equations or keep (3.13) and calculate the optimal 12 elements of T under the non-linear constraints imposed by the underlying 10 independent parameters.

Neither approach is practical and the usual compromise is to look for the optimal solution for the T_{mn}'s neglecting the constraints among them. To accomplish this, let's assume that we are using the scene points P_l, $l = 1,...,N$, with coordinates in the world frame (x_l, y_l, z_l), for camera calibration. Let their corresponding image points be denoted by the pairs (i_l, j_l). For each pair of the corresponding scene point and the image point, we can write the following two equations derived from (3.10):

$$i_l = \frac{T_{11}\,x_l + T_{12}\,y_l + T_{13}\,z_l + T_{14}}{T_{31}\,x_l + T_{32}\,y_l + T_{33}\,z_l + T_{34}} \qquad (3.21a)$$

$$j_l = \frac{T_{21}\,x_l + T_{22}\,y_l + T_{23}\,z_l + T_{24}}{T_{31}\,x_l + T_{32}\,y_l + T_{33}\,z_l + T_{34}} \tag{3.21b}$$

Since the equations will not be altered if all the T_{mn} are scaled by an arbitrary factor, we scale them by T_{34} implicitly by setting T_{34} to 1. These two equations can be written as

$$T_{11}\,x_l + T_{12}\,y_l + T_{13}\,z_l + T_{14} - T_{31}\,i_l\,x_l - T_{32}\,i_l\,y_l - T_{33}\,i_l\,z_l = i_l \tag{3.22a}$$

$$T_{21}\,x_l + T_{22}\,y_l + T_{23}\,z_l + T_{24} - T_{31}\,j_l\,x_l - T_{32}\,j_l\,y_l - T_{33}\,j_l\,z_l = j_l \tag{3.22b}$$

for l=1,2,...,N. For all l collectively, these equations can be expressed in a matrix form if we define a vector U of unknowns and a right-hand-side vector B as

$$U = \begin{bmatrix} T_{11} \\ T_{12} \\ T_{13} \\ T_{14} \\ T_{21} \\ T_{22} \\ T_{23} \\ T_{24} \\ T_{31} \\ T_{32} \\ T_{33} \end{bmatrix} \qquad B = \begin{bmatrix} i_1 \\ i_2 \\ . \\ . \\ i_N \\ j_1 \\ j_2 \\ . \\ . \\ j_N \end{bmatrix} \tag{3.23}$$

Then the equations (3.22) for l = 1,...,N can be expressed as

$$A\,U = B \tag{3.24}$$

where A is the matrix of coefficients; the elements of A consist of the world coordinates of the scene points used for calibration.

This is an overdetermined system of linear equations for N greater than 5, since for N equal to 6 we will have 12 equations for only 11 unknowns. The system of equations will have a solution only if the columns of A are linearly independent. It can be seen from Eq. (3.22) that linear independence implies that the points P_l should not be coplanar for otherwise any of the columns of A involving one of the world coordinates could be written as a linear combination of the other two.

Due to inconsistencies among the equations caused by measurement and digitization errors, round-off, etc., there is no exact solution for (3.24) with N greater than 5. Consequently we will find a *best possible solution* in the least squares sense. This is a

solution that minimizes the expression

$$\left[A\,U - B\right]^T \left[A\,U - B\right] \tag{3.25}$$

which is the solution of the *normal equations:*

$$A^T A\,U = A^T B \tag{3.26}$$

A commonly used solution to Eqs. (3.25) and (3.26) is the pseudo-inverse solution and is given by

$$U = (A^T A)^{-1} A^T B \tag{3.27}$$

Note that, at least theoretically, this solution is guaranteed to exist since when the columns of A are linearly independent the inverse of $A^T A$ will always exist. Such pseudo-inverse solutions are frequently used to solve overdetermined sets of equations and the solutions produced are generally acceptable whenever the data are not excessively corrupted by noise. We have chosen to use the following method which we believe yields superior results in the presence of noise and errors in the elements of the matrix A. In the method we use, the solution to Eq. (3.24) is found by first computing the QR decomposition of A obtained by using Housholder transformations and then solving, by using the pseudo-inverse method, the equivalent system $R\,P^T\,U = Q^T\,B$, where P is a permutation matrix, Q an orthogonal matrix, and R an upper triangular matrix. The solution so obtained is then used as an initial guess for a routine that computes the least squares optimal solution of the set of equations (3.21) with $l = 1,...,N$, using a modified Levenger-Marquardt method. This procedure is summarized in Fig. 3.4.

3.3 CALIBRATION METHOD BASED ON PINHOLE MODEL INCORPORATING RADIAL DISTORTION

If one must use the pinhole model, the calibration methods that take into account radial lens distortion are capable of producing higher accuracy results. To study the potential of this approach for our application, we implemented a procedure first advanced by Tsai for this augmentation of the pinhole method. We chose his method for implementation for the reason that it allows the world points used for calibration to be coplanar -- a great convenience for calibrating the cameras mounted on a mobile robot, since all the calibration points can be marked on a wall or some other flat plane. As was mentioned in the preceding section, the previous calibration method does not admit coplanar points.

Input: $P_{W_l} = (x_l, y_l, z_l)$; $P_{I_l} = (i_l, j_l)$ $l = 1, \cdots, N$

Output: T_{mn} $i = 1,2,3$; $j = 1,2,3,4$

1: Solve the normal equation:

$$A^T A\, U = A^T B$$

2: Solve the non linear equations for an optimal solution using U as an initial guess:

$$i_l = \frac{T_{11}\, x_l + T_{12}\, y_l + T_{13}\, z_l + T_{14}}{T_{31}\, x_l + T_{32}\, y_l + T_{33}\, z_l + T_{34}}$$

$$j_l = \frac{T_{21}\, x_l + T_{22}\, y_l + T_{23}\, z_l + T_{24}}{T_{31}\, x_l + T_{32}\, y_l + T_{33}\, z_l + T_{34}}$$

$$l = 1, \cdots, N$$

Figure 3.4: Summary of the procedure used to compute the calibration matrix.

3.3.1 Camera Model

This subsection describes the camera model, defines the calibration parameters and shows the derivation of the equations that represent the imaging process. Fig. 3.5 illustrates the basic geometry of the camera model. O represents the center of projection or the focal point, O' is the image center or point where the optic axis (in our case the Z axis) pierces the image plane and P_u is the image of a scene point P for a perfect pinhole camera. P_d is the actual image of the scene point P that differs from P_u due to lens distortion. The distance from the image plane to the center of projection (length of the segment $\overline{OO'}$) is f. The system centered at O_w represents the world coordinate frame and the one centered at O is the camera coordinate frame.

The transformation from the world coordinates (x_w, y_w, z_w) of a scene point P to the coordinates (x_f, y_f) of the corresponding image point can be obtained by performing the following steps. First, we compute the coordinates of P in the camera frame centered at O'. We will represent these coordinates by (x, y, z).

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} + T \tag{3.28}$$

where R is the 3×3 rotation matrix

$$R = \begin{bmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \\ r_7 & r_8 & r_9 \end{bmatrix} \tag{3.29}$$

and T is the translation vector

$$T = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} \tag{3.30}$$

Note that not all r_i are independent since a rotation can be uniquely specified by three independent parameters, such as Euler angles (these are the pan, tilt and swing angles).

The next step is to relate the coordinates (x, y, z) of P in the camera coordinate system to the coordinates (x_u, y_u) of its ideal pinhole image at P_u. By comparing similar triangles in Fig. 3.5 we can obtain

$$x_u = f \frac{x}{z} \tag{3.31a}$$

$$y_u = f \frac{y}{z} \tag{3.31b}$$

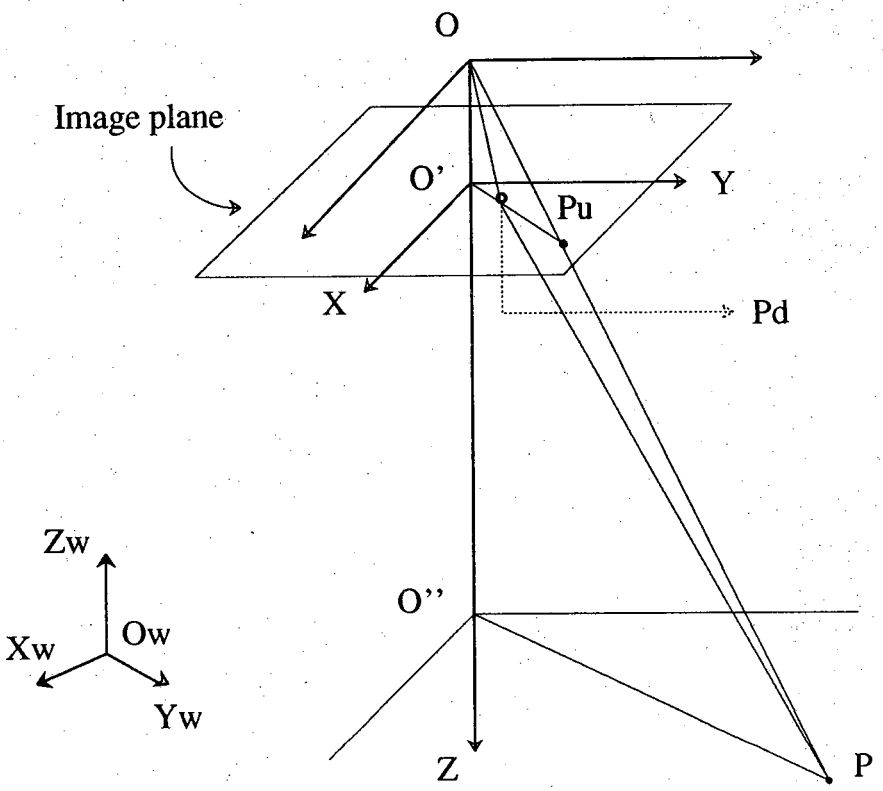


Figure 3.5: Illustration of the radial alignment constraint. P_u and P_d represent respectively the pinhole image and the distorted (actual) image of the object point P. Observe that $OP_u \| OP_d \| O'P$.

The third step is to compute the coordinates (x_d, y_d) of the actual image point P_d which takes into account the lens distortion. There are two types of distortions, radial and tangential. Each of them can modeled by an infinite series [3] but for computer vision applications only the first terms of the radial series need to be considered. The distortion in the location of P_d accounts for only the radial component. Therefore we can write

$$x_d + D_x = x_u \tag{3.32a}$$

$$y_d + D_y = y_u \tag{3.32b}$$

with

$$\begin{aligned} D_x &= x_d\,(k_1\,r^2 + k_2\,r^4 + \ldots) \\ D_y &= y_d\,(k_1\,r^2 + k_2\,r^4 + \ldots) \\ r &= \sqrt{x_d^2 + y_d^2} \end{aligned} \tag{3.33}$$

If we combine equations (3.28), (3.31) and (3.32) we can write

$$x_d + D_x = f\,\frac{r_1\,x_w + r_2\,y_w + r_3\,z_w + T_x}{r_7\,x_w + r_8\,y_w + r_9\,z_w + T_z} \tag{3.34a}$$

$$y_d + D_y = f\,\frac{r_4\,x_w + r_5\,y_w + r_6\,z_w + T_x}{r_7\,x_w + r_8\,y_w + r_9\,z_w + T_z} \tag{3.34b}$$

Following Tsai, for the distortion terms D_x and D_y we have only shown two terms in the series for each. The terms contain lens-specific coefficients k_1 and k_2. The calibration procedure presented by Tsai and reimplemented by us computes these coefficients.

But, of course, (x_d, y_d) are not the final image coordinates. The image coordinates (x_f, y_f) are obtained after digitizing the image, and, also, the origin for the digitized representation will usually be at a corner of the image, as opposed to being at the center. If we assume that the x direction in the digitized image corresponds to the scan line direction, the equations that relate (x_d, y_d) and (x_f, y_f) are:

$$x_f = s_x\,d_x'^{-1}\,x_d + C_x \tag{3.35a}$$

$$y_f = d_y^{-1}\,y_d + C_y \tag{3.35b}$$

$$d'_x = d_x\,\frac{N_{cx}}{N_{fx}} \tag{3.36}$$

where C_x and C_y are the row and column indices of the center of the digitized frame in relation to the center of the analog image, and d_x and d_y are the distances between the centers of the adjacent pixels in the x and y directions, respectively. N_{cx} is the number of sensors, each sensor corresponding to a pixel, in the x direction and N_{fx} is the number of scan lines, or, equivalently, the number of pixels in the y direction. (The x direction is assumed to be the scan line direction.) The quantity s_x is the so-called uncertainity

image scale factor, which accounts for possible timing mismatches between the camera scanning hardware and the image acquisition hardware or the errors in timing of the TV scanning itself. Tsai has reported that even an one percent difference between the timings can cause up to five pixel error in the image resolution.

When a CCD camera is used, the parameters d_x, d_y, N_{cx} and N_{fx} can be obtained from the information supplied by the manufacturer. If a vidicon camera is used instead, some of these parameters are not known a priori and this calibration technique can not be used. However, in this case one may use a multi-plane calibration method introduced by Tsai; this method exploits the fact that all the missing information can be grouped into only one additional unknown. Note that the product $s_x d'_x{}^{-1}$ can be treated as a single parameter and that d_y can be set to one since the focal length scales the image in both the x and y directions. If we do this the computed focal length f will be a product of the actual focal length and the scale factor in y.

The process of obtaining the image coordinates from world coordinates is summarized in Fig. 3.6.

The parameters used in the camera model discussed so far in this subsection can be categorized into two classes: *intrinsic parameters* and *extrinsic parameters*. The extrinsic parameters describe the position and orientation of the camera in the world coordinate system and the intrinsic parameters determine the image forming process. Extrinsic parameters are the elements of the rotation matrix R and the components of the translation vector T. The other parameters, N_{cx}, N_{fx}, d_x, d_y, f, k_1, k_2, s_x, C_x and C_y, are the intrinsic parameters.

The procedure we describe in the following subsection is sufficient to determine all the extrinsic parameters and only f, k_1 and k_2 of the intrinsic parameters. N_{cx}, N_{fx}, d_x and d_y are assumed to be known at the time of calibration and C_x, C_y and s_x are not calibrated, meaning no information is calculated on these parameters. Tsai has suggested that (C_x, C_y) be set to the center pixel of the digitized image and has proposed several techniques for computing s_x.

3.3.2 Calibration Procedure

The method consists of two stages. In the first stage of the procedure, some of the parameters are found using linear equations. These linear equations represent physical constraints on the locations of the object points and their images, each constraint saying that, if only radial distortion is present, the image of an object point must lie on a line that is the projection on the image plane of a line joining the object point to the focal center, as shown in Fig. 3.5; such constraints are referred to as *radial alignment*

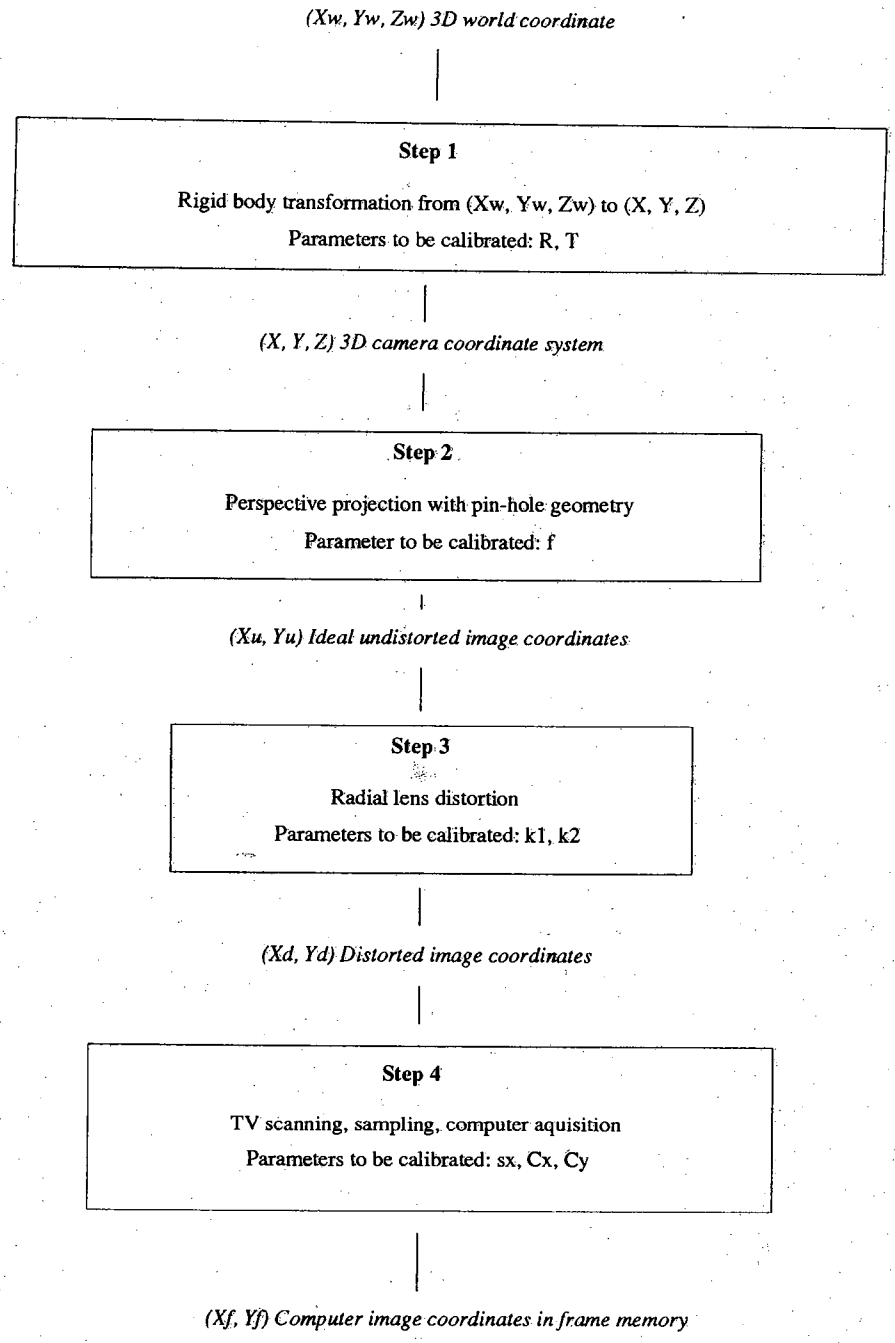


Figure 3.6: The four steps in the transformation from 3D world coordinates to computer image coordinates and the parameters involved.

constraints. These constraints allow us to easily compute some of the elements of the rotation matrix and a product of two of the components of the translation vector. Subsequently, during the first phase computations, the rest of the elements of the rotation matrix and two of the components of the translation vector are computed by enforcing the orthonormality property of the rotation matrix.

In the second stage, we start out by first estimating approximations to the remaining parameters by using the pinhole projective equations (with distortion terms set to zero). The exact solution for these parameters is then computed using the full equations and any standard optimization scheme. We will now elaborate on the two stages.

a) First Stage In the first stage, we write a direct relationship between the actual image coordinates (x_d, y_d) and the world coordinates (x_w, y_w, z_w). We do so by recognizing that under the radial alignment constraint, the direction of the vector $\overrightarrow{O''P}$ is radially aligned with the vector $\overrightarrow{O'P_d}$ (Fig. 3.5). Therefore, if the vector $\overrightarrow{O''P}$ is expressed in the camera coordinate frame centered at O', we can write

$$\frac{O''P_x}{O''P_y} = \frac{x_d}{y_d} \tag{3.37}$$

However, $(O''P_x\ ,\ O''P_y)$, both measured in the camera coordinate frame, are related to the (x_w, y_w, z_w) measured in the world frame through equations (3.28). Therefore,

$$O''P_x = r_1 x_w + r_2 y_w + r_3 z_w + T_x \tag{3.38a}$$

$$O''P_y = r_4 x_w + r_5 y_w + r_6 z_w + T_y \tag{3.38b}$$

Combining equations (3.37) and (3.38), we get

$$\frac{r_1 x_w + r_2 y_w + r_3 z_w + T_x}{r_4 x_w + r_5 y_w + r_6 z_w + T_y} = \frac{x_d}{y_d} \tag{3.39}$$

Recasting (3.39) in a more convenient form we obtain

$$\begin{bmatrix} y_d x_w & y_d y_w & y_d z_w & y_d & -x_d x_w & -x_d y_w & -x_d z_w \end{bmatrix} \begin{bmatrix} T_y^{-1} r_1 \\ T_y^{-1} r_2 \\ T_y^{-1} r_3 \\ T_y^{-1} T_x \\ T_y^{-1} r_4 \\ T_y^{-1} r_5 \\ T_y^{-1} r_6 \end{bmatrix} = x_d \tag{3.40}$$

Equation (3.40) corresponds to a single scene point. Since we have seven unknowns, if the number of scene points, N, is greater than 7, we can establish an

overdetermined set of linear equations of the form $A\ T = X$, where A is an $N \times 7$ matrix of coefficients, X is composed of x_d's appearing on the right hand side of equations like (3.40), and T is the vector of the unknowns shown as a column vector in equation (3.40). However, note that if the scene points are coplanar, some of the columns in A will become linearly dependent and the set of equations will have no unique solution.

If the world coordinate system is selected such that the plane $z_w = 0$ coincides with the calibration plane and the origin of the world system is not close to the camera y-axes then equation (3.40) can be rewritten in terms of five unknowns as

$$\begin{bmatrix} y_d x_w & y_d y_w & y_d & -x_d x_w & -x_d y_w \end{bmatrix} \begin{bmatrix} T_y^{-1} r_1 \\ T_y^{-1} r_2 \\ T_y^{-1} T_x \\ T_y^{-1} r_4 \\ T_y^{-1} r_5 \end{bmatrix} = x_d \tag{3.41}$$

and the resulting system of equations can now be solved for the five unknowns $T_y^{-1} r_1$, $T_y^{-1} r_2$, $T_y^{-1} T_x$, $T_y^{-1} r_4$ and $T_y^{-1} r_5$, assuming that the calibration points are not collinear. A complete discussion on the existence and uniqueness of this solution can be found in [9]. The reason for selecting the origin of the world frame far from the camera y-axes is to avoid having $T_y = 0$. Note that the origin and orientation of the world coordinate frame can be chosen as desired since they are under user control.

With the procedure outlined so far, we now have the means to compute the quantities $T_y^{-1} r_1$, $T_y^{-1} r_2$, $T_y^{-1} T_x$, $T_y^{-1} r_4$ and $T_y^{-1} r_5$. In the next step in the first stage, all the elements of the rotation matrix R, and T_x and T_y will be determined using the orthonormality constraints that must be satisfied by the rows and columns of R. To see how this can be done, let's first define a 2×2 matrix C as

$$C \equiv \begin{bmatrix} r'_1 & r'_2 \\ r'_4 & r'_5 \end{bmatrix} \equiv \begin{bmatrix} \frac{r_1}{T_y} & \frac{r_2}{T_y} \\ \frac{r_4}{T_y} & \frac{r_5}{T_y} \end{bmatrix} \tag{3.42}$$

C is the upper 2×2 submatrix of R scaled by the factor T_y^{-1}. The following lemma puts a restriction on how one can scale the 2×2 submatrix of a 3×3 orthonormal matrix.

Lemma I: There do not exist two 3×3 orthonormal matrices that differ in their 2×2 submatrix by a scale factor other than ± 1.

The lemma implies that if a 2×2 submatrix of an orthonormal 3×3 matrix is given except for a scale factor, then the scale factor is unique except for its sign. Using this

property, if an entire row or column of C does not vanish, the square of the scale factor (in our case T_y) can be determined by using

$$T_y^2 = \frac{S_r - [S_r^2 - 4(r'_1 r'_5 - r'_4 r'_2)^2]^{1/2}}{2(r'_1 r'_5 - r'_4 r'_2)^2} \tag{3.43}$$

where $S_r = r'^2_1 + r'^2_2 + r'^2_4 + r'^2_5$. Clearly, this equation can not be used when the denominator on the right hand side vanishes. In that case, T_y^2 can be computed from

$$T_y^2 = (r'^2_i + r_j'^2)^{-1} \tag{3.44}$$

where r'_i and r'_j are the elements of C that do not vanish. A detailed proof of Lemma I and derivation of Eqs. (3.43) and (3.44) can be found in [9].

So, now we have the means to also compute the magnitude $|T_y|$; the next step naturally is to determine its sign. In order to do this, we start out by assuming that T_y is positive. Using this positive value, we can determine all the unknowns (r_1, r_2, r_4, r_5, T_x and T_y) from equations like (3.39). Now if we select an object point whose image coordinates (X, Y) are away from the image center and substitute its world coordinates in Eq. (3.39), then the result we obtain for X may or may not agree in its sign with the actual sign of X. In the first case T_y was indeed positive, however, in the second case, we have to reverse its sign.

Now we can compute r_1, r_2, r_4, r_5, T_x and T_y. Subsequently, during the first stage computations, we again use the orthonormal and right handed property of R to determine r_3, r_6, r_7, r_8 and r_9 as follows. First, r_3 is calculated from r_1 and r_2 by using $r_3 = +(1 - r_1^2 - r_2^2)^{1/2}$. Similarly, we have $r_6 = s(1 - r_4^2 - r_5^2)^{1/2}$, with $s = -sgn(r_1 r_4 + r_2 r_5)$. And, finally, r_7, r_8 and r_9 are determined by using the vector product of the first two rows of R. However, this is not the only possible solution for R, since we could have chosen r_3 negative and that would cause a sign change for r_6, r_7 and r_8. We can arbitrarily select one of the two solutions of R. We will see that if the wrong choice is made then in the second stage when the focal length f is determined it will be negative, indicating that the signs of r_3, r_6, r_7 and r_8 should be reversed.

b) Second Stage. In this stage, we first estimate approximations to the parameters f and Tz ignoring lens distortion, then a more accurate solution is found taking into account the radial lens distortion and using a nonlinear optimization method. The latter step, involving nonlinear optimization, also yields values for the parameters k_1 and k_2 of the radial lens distortion (see Eqs. (3.33)).

Initially ignoring the radial lens distortion implies that we can set D_x and D_y to zero in equations (3.32). Therefore, equations (3.34) reduce to the following simpler forms involving the unknowns f and T_z:

$$x_d = f\frac{r_1 x_w + r_2 y_w + r_3 z_w + T_x}{r_7 x_w + r_8 y_w + r_9 z_w + T_z} \tag{3.45a}$$

$$y_d = f\frac{r_4 x_w + r_5 y_w + r_6 z_w + T_y}{r_7 x_w + r_8 y_w + r_9 z_w + T_z} \tag{3.45b}$$

These two equations can be recast into the following forms for computer implementation:

$$\begin{bmatrix} x & -x_d \end{bmatrix} \begin{bmatrix} f \\ T_z \end{bmatrix} = w\, x_d \tag{3.46a}$$

$$\begin{bmatrix} y & -y_d \end{bmatrix} \begin{bmatrix} f \\ T_z \end{bmatrix} = w\, y_d \tag{3.46b}$$

where

$$\begin{aligned} x &= r_1 x_w + r_2 y_w + r_3 z_w + T_x \\ y &= r_4 x_w + r_5 y_w + r_6 z_w + T_y \\ w &= r_7 x_w + r_8 y_w + r_9 z_w + T_z \end{aligned} \tag{3.47}$$

Since we are using a coplanar set (but they cannot be colinear) of calibration points lying on the plane $z_w = 0$, equations (3.47) reduce to

$$\begin{aligned} x &= r_1 x_w + r_2 y_w + T_x \\ y &= r_4 x_w + r_5 y_w + T_y \\ w &= r_7 x_w + r_8 y_w + T_z \end{aligned} \tag{3.48}$$

With several calibration points, either equation (3.46b) or (3.46b) (or both combined) yield an overdetermined system of linear equations that can be used to solve for f and T_z. This can be accomplished by using the same procedure as outlined in Section 3.2.2 if the calibration plane has not been chosen exactly parallel to the image plane, otherwise the system formed becomes linearly dependent.

Note that from the set of elements of R whose sign we had to guess in the first stage, only r_7 and r_8 appear in w in equations (3.46) and that a wrong choice for their signs will result in a negative focal length. Therefore, it is at this point that we can reverse their signs if necessary.

The only step left is to revise our estimates for f and T_z taking the radial distortion into account and to determine the values for the parameters k_1 and k_2. The final solutions for f and T_z and the values for k_1 and k_2 are found simultaneously from the complete model equations (3.34) via a nonlinear optimization that is started with the estimates for f and T_z and with $k_1 = 0$ and $k_2 = 0$. The complete calibration procedure is summarized in Fig. 3.7.

Stage 1

1: Convert from computer coordinates (x_f, y_f) to distorted image coordinates (x_d, y_d).

2: Compute $T_y^{-1}r_1, T_y^{-1}r_2, T_y^{-1}T_x, T_y^{-1}r_4$ from the radial alingment constraint equations.

3: Compute $(r_1, r_2, r_4, r_5, T_x, T_y)$ from $(T_y^{-1}r_1, T_y^{-1}r_2, T_y^{-1}T_x, T_y^{-1}r_4)$ using the orthonormal property of R.

Stage 2

4: Compute a first approximation to T_z and f using the perspective projection equations ignoring lens distortion.

5: Compute the final value of T_z, f, k_1 and k_2 using the full model equations.

Figure 3.7: The two stage calibration procedure.

3.4 THE TWO-PLANE METHOD FOR CAMERA CALIBRATION

In general, the purpose of camera calibration is to solve one or both of the following two problems:

- The projection problem: given the location of a point in space, predict the location of its image on the image plane.
- The back-projection problem: given location on the image plane of the image of a point, compute the line of sight on which the point being imaged must lie.

Martins et al. [11] presented the two-plane calibration technique for solving the back-projection problem. This technique provides exactly the information needed, this is the line in space that corresponds to the line of sight of a given pixel, without any explicit camera model.

3.4.1 Camera Model

In this method, instead of describing a camera model by the usual parameters such as the focal length, location and orientation of the imaging plane, etc., and attempting to determine these parameters through calibration, relationships are established between the coordinates of scene points and the coordinates of the corresponding image points for scene points in two different planes, called the calibration planes. Once these relationships are determined, then for an image point I corresponding to an arbitrary scene point P, we can find, using the inverse of the relationships, the points on each of the calibration planes that would give rise to the image point I. A line joining the two points on the two calibration planes then defines a line-of-sight to the scene point P.

3.4.2 Calibration Procedure

Martins et al. have proposed using three different types of relationship between the scene points on the calibration planes and their corresponding image points: linear, quadratic, and linear spline. Although we implemented the quadratic approach only, we will briefly describe all three of them.

The Case of Linear Relationship:

The camera is shown a set of illuminated scene points located on what is called the calibration plane #1 and their corresponding image points are recorded. A linear relationship is expressed between the scene points and their corresponding image points.

This process is repeated for the second calibration plane. Let $P_i = (X_i, Y_i, Z_i)$ denote a typical point on the i^{th} calibration plane and let $L_i = (row,\ col,\ 1)$ denote its image point. Let a 3×3 matrix A_i express the linear relationship between P_i and L_i:

$$P_i = A_i L; \qquad i = 1, 2, \tag{3.49}$$

Note that each calibration plane is characterized by a single matrix A_i.

Clearly, with more than three points per plane a set of overdetermined linear equations can be established and solved for the elements of A_i using the method described in Section 3.2.2. As a result, we can compute A_1 and A_2 for the calibration planes #1 and #2, respectively. That is the end of camera calibration. Now when an arbitrary scene point is shown to the camera, from its image point we can compute, by using A_1 and A_2, two points -- let's call them P'_1 and P'_2 -- which would be the intersection of the line-of-sight to the arbitrary scene point with the two calibration planes. Thus we can find the line-of-sight corresponding to a given scene point after the calibration phase.

The Case of Quadratic Relationship:

The quadratic interpolation is similar to the linear case except that, for each calibration plane, the relationship between a scene point P_i and its corresponding image point L_i is expressed by

$$P_i = A_i Q_i; \quad i = 1, 2, \tag{3.50}$$

where, as before, $P_i = (X_i, Y_i, Z_i)$, but where the vector Q_i is given by $Q_i = (row^2, col^2, row\ col, row, col, 1)$. In other words, we now have nonlinear relationship between a scene point and its image; this is an attempt at capturing the nonlinearities on a lens. Note that for each calibration plane, A_i is now a 6×6 matrix. Also note that each calibration plane is characterized by a single matrix A_i.

Calibration consists of showing to the camera a set of illuminated scene points, at least six, in each calibration plane, each illuminated scene point and its image being related by (3.50). Given a sufficient number of points in each plane, we can find the matrix A_i for that plane. Thus we can find A_1 and A_2 for the two calibration planes. That completes the calibration of the camera. Now if we show an arbitrary scene point to the camera and record its image point. Then, using A_1 and A_2, we can find the two points -- let's call them P'_1 and P'_2 -- where the line-of-sight to the scene point intersects the two calibration planes. The points P'_1 and P'_2 thus obtained then define the line-of-sight.

Using Linear Splines:

In both the linear and the quadratic interpolation, the approach is to globally fit, for each calibration plane separately, a function describing the relationship between the scene points and their image locations. In other words, the calibration plane #i was characterized by a single matrix A_i, meaning that, for the calibration plane #i, the relationship *everywhere* between the coordinates of a scene point and its image point was described by the same matrix A_i. Such global characterization has the effect of averaging the errors over all the pixels corresponding to a calibration plane so the resultant line-of-sight may not be exact for any pixel but close for all. For some cases, depending on the nature and extent of lens distortion, such global characterization may not be appropriate.

In some cases, a better approach may be to use separate functions, each valid only over a small and localized region of the image plane, for describing the relationships between the scene points on a calibration plane and the corresponding image points. One easy way to accomplish this is by the use of linear splines; these are linear functions, each valid over a small approximately triangular region of the image plane, for describing the relationship between the scene points on a calibration plane and the corresponding image points. Each calibration plane is divided into triangular tessellations, the vertices of the triangles form the illuminated scene points that are shown to the camera for calibration. The coordinates of the vertices of each triangle and the corresponding image coordinates are used to compute a linear relationship, as in the "Linear Case" discussed above, between the scene points that would lie within the triangle and their corresponding image points. Therefore, for the j^{th} triangle in the i^{th} calibration plane, the relationship between the coordinates P_i of a scene point and its image L_i are given by

$$P_i = A_{ij} L; \quad i = 1, 2; \; j = 1, \cdots, N, \tag{3.51}$$

where L, P and A_{ij} are as in (3.49), the difference being that now there is one matrix A_{ij} for each of the N triangles. An exact solution is now obtained for each triangle.

3.5 DETERMINATION OF THE PHYSICAL PARAMETERS OF THE CAMERA

In Section 3.2, we discussed a calibration method that was based on the ideal pinhole assumption. The processing discussed in Section 3.2.2 yields a 3×4 calibration matrix T whose elements are displayed in Eq. (3.10). After the matrix T is computed, one is still faced with the problem of having to compute the actual camera parameters. In this section, we will describe a procedure, which was originally proposed by Ganapathy [6], for computing the parameters of the camera from the elements of the T matrix.

Of course, the reader could ask: Is it really necessary to compute the physical parameters of the camera, such as its focal length; could we not directly use the matrix T? It is, of course, true that the elements of T can be used directly to compute the line-of-sight corresponding to any pixel in the image by using Eq. (3.20). It is also true that given, say, a CAD representation of the hallways, from the elements of T one could write a routine to construct an image that a camera at a given position and with a given view-vector would see. However, many CAD rendering programs, especially of the ray-tracing variety, explicitly require parameters such as the focal length, the direction of the view-vector, etc.* As will be discussed in Chapter 6, a rendering program is used to generate scene expectation maps from CAD models for vision-guided navigation using the PSEIKI system [15].

From Eqs. (3.10), (3.6a) and (3.6b), we can show that the elements of T are related to the physical parameters of a camera by the following relationships:

$$T_{11} = \frac{\frac{f}{\Delta u} h_x + i_0 a_x}{-\vec{C}\cdot\hat{a}} \tag{3.52a}$$

$$T_{12} = \frac{\frac{f}{\Delta u} h_y + i_0 a_y}{-\vec{C}\cdot\hat{a}} \tag{3.52b}$$

$$T_{13} = \frac{\frac{f}{\Delta u} h_z + i_0 a_z}{-\vec{C}\cdot\hat{a}} \tag{3.52c}$$

$$T_{14} = -\frac{\frac{f}{\Delta u} \vec{C}\cdot\hat{h} + i_0 \vec{C}\cdot\hat{a}}{-\vec{C}\cdot\hat{a}} \tag{3.52d}$$

$$T_{21} = \frac{\frac{f}{\Delta v} v_x + j_0 a_x}{-\vec{C}\cdot\hat{a}} \tag{3.52e}$$

$$T_{22} = \frac{\frac{f}{\Delta v} v_y + j_0 a_y}{-\vec{C}\cdot\hat{a}} \tag{3.52f}$$

* A ray tracing program for rendering an image from a CAD representation 'shoots' rays from the center of projection and through the image plane. By computing the intersection of a given ray with the surfaces of the scene, the ray-tracing program figures out how to render the scene and how to carry out hidden surface removal. Clearly then, in order to 'shoot' the rays, the program needs to know the distance from the center of projection to the image plane, which is the focal length of the camera, the view-vector associated with the camera, etc.

$$T_{23} = \frac{\frac{f}{\Delta v} v_z + j_0 a_z}{-\vec{C}\cdot\hat{a}} \tag{3.52g}$$

$$T_{24} = -\frac{\frac{f}{\Delta v} \vec{C}\cdot\hat{v} + j_0 \vec{C}\cdot\hat{a}}{-\vec{C}\cdot\hat{a}} \tag{3.52h}$$

$$T_{31} = \frac{a_x}{-\vec{C}\cdot\hat{a}} \tag{3.52i}$$

$$T_{32} = \frac{a_y}{-\vec{C}\cdot\hat{a}} \tag{3.52j}$$

$$T_{33} = \frac{a_z}{-\vec{C}\cdot\hat{a}} \tag{3.52k}$$

$$T_{34} = 1 \tag{3.52l}$$

In these expressions, we have scaled all the T_{ij}'s by T_{34}, which is consistent with the assumption made in Section 3.2.2. From Eq. (3.10), $T_z = -C'_z$. However, from Eq. (3.12) and (3.13), $C'_z = RC = -\vec{C}\cdot\hat{a}$.

We will now describe Ganapathy's procedure for processing the above equations for computing the parameters $\vec{C}$, $\hat{h}$, $\hat{v}$, $\hat{a}$, i_0, j_0, $\frac{f}{\Delta u}$ and $\frac{f}{\Delta v}$. We will use the symbols T_1, T_2 and T_3 to denote the first, second and third rows of the leftmost 3×3 submatrix of the calibration matrix T, respectively, We obtain from Eqs. (3.52a), (3.52b) and (3.52c):

$$T_1\cdot T_1 = T_{11}^2 + T_{12}^2 + T_{13}^2 = \left[\frac{f/\Delta u}{-\vec{C}\cdot\hat{a}}\right]^2 + \left[\frac{i_0}{-\vec{C}\cdot\hat{a}}\right]^2 \tag{3.53}$$

by using the fact that $\hat{h}$, $\hat{v}$ and $\hat{a}$ are a mutually orthogonal set of unit vectors. Similarly,

$$T_2\cdot T_2 = T_{21}^2 + T_{22}^2 + T_{23}^2 = \left[\frac{f/\Delta v}{-\vec{C}\cdot\hat{a}}\right]^2 + \left[\frac{j_0}{-\vec{C}\cdot\hat{a}}\right]^2 \tag{3.54}$$

$$T_3\cdot T_3 = T_{31}^2 + T_{32}^2 + T_{33}^2 = \frac{1}{\left[-\vec{C}\cdot\hat{a}\right]^2} \tag{3.55}$$

$$T_1\cdot T_3 = T_{11}T_{31} + T_{12}T_{32} + T_{13}T_{33} = \frac{i_0}{\left[-\vec{C}\cdot\hat{a}\right]^2} \tag{3.56}$$

and

$$T_2\cdot T_3 = T_{21}T_{31} + T_{22}T_{32} + T_{23}T_{33} = \frac{j_0}{\left[-\vec{C}\cdot\hat{a}\right]^2} \tag{3.57}$$

These equations are suggestive of a solution strategy for the parameters. From Eq. (3.55), we can directly compute $-\vec{C}\cdot\hat{a}^2$. Subsequently, from Eqs. (3.56) and (3.57), we can compute i_0 and j_0. Following that, Eqs. (3.53) and (3.54) yield the magnitudes of $\frac{f}{\Delta u}$ and $\frac{f}{\Delta v}$.

Note that the signs of $-\vec{C}\cdot\hat{a}$, $\frac{f}{\Delta u}$ and $\frac{f}{\Delta v}$ are closely related. We can arbitrarily choose $-\vec{C}\cdot\hat{a}$ and $\frac{f}{\Delta u}$ to be positive and then determine the sign of $\frac{f}{\Delta v}$. The signs of $\frac{f}{\Delta u}$ and $\frac{f}{\Delta v}$ depend on the polarity of the axes of the camera coordinate frame, meaning the directions in which the row and column numbers of the pixels (i, j) increase. The quantity $\frac{f}{\Delta u}$ can always be assumed positive. Then the sign of $\frac{f}{\Delta v}$ depends on the on the polarity of the I–J axes. Note that the u–v axes of the camera coordinate system were defined in Section 3.2.1 as right-handed and parallel to the directions of the rows and columns of the computer image respectively. However, if the columns are counted from left to right and the rows from the upper to the lower border of the picture, then this is equivalent to inverting the direction of the vector $\hat{v}$ or to have the quantity $\frac{f}{\Delta v}$ negative.

The sign of $-\vec{C}\cdot\hat{a}$ can be reversed if necessary by changing the direction of the view vector, $\hat{a}$, by 180°. Then in order to keep the coordinate system defined by $(\hat{h}, \hat{v}, \hat{a})$ right-handed, we need to also reverse the direction of $\hat{v}$, which amounts to changing the sign of $\frac{f}{\Delta v}$.

Once we have assumed the signs of $\frac{f}{\Delta u}$ and $-\vec{C}\cdot\hat{a}$ to be positive, the sign of $\frac{f}{\Delta v}$ can be determined as follows. Assume for a moment that $\frac{f}{\Delta v}$ is positive. Since at this point the values of $-\vec{C}\cdot\hat{a}$, $\frac{f}{\Delta u}$, i_0, a_x, a_x, a_y and a_z and $\hat{a}$ are known, u_x, u_y, u_z, can be obtained from equations (3.52a), (3.52b) and (3.52c), respectively. Similarly, from the expressions for T_{21}, T_{22} and T_{23} we can determine the values for v_x, v_y and v_z respectively, giving us the vector $\hat{v}$. The vector $\hat{v}$ can also be obtained from $\hat{v} = \hat{a}\times\hat{h}$. If both the solutions for $\hat{v}$ coincide, the choice of sign for $\frac{f}{\Delta v}$ was correct otherwise it must be reversed.

With the implementation described so far, the only quantities left to be computed are the components of the vector $\vec{C}$ f, Δu and Δv. The components of $\vec{C}$ can be recovered from equations (3.52d) and (3.52h). The focal length f can be obtained if we have knowledge of the size of an image pixel, that is Δu and Δv; usually, this information is

available from the manufacturer of the camera.

If the procedure described in Section 3.2.2 is used to compute the calibration matrix, T, the assumption of orthonormality of the vectors $\hat{h}$, $\hat{v}$ and $\hat{a}$ may not be satisfied strictly since the calibration procedure does not enforce this constraint explicitly. Ganapathy has proposed inclusion of an additional parameter δ to measure the extent to which the calibration matrix approximates the true perspective projection matrix.

Let us define three mutually orthogonal unit vectors $\hat{h}'$, $\hat{v}'$ and $\hat{a}'$. Let us assume that $\hat{h}$ and $\hat{v}$ are not exactly perpendicular to each other but slightly off by an angle δ (see Fig. 3.8). If we assume $\hat{h}'$ and $\hat{a}'$ coincide with $\hat{h}$ and $\hat{a}$, respectively, as in Fig. 3.8, then the coordinates of $\hat{v}$ in the $\hat{h}'$, $\hat{v}'$, $\hat{a}'$ system are given by

$$\begin{aligned} v_1 &= \sin\delta, \\ v_2 &= \cos\delta, \\ v_3 &= 0 \end{aligned} \tag{3.57}$$

In order to compute δ, we can use the fact that $T_1 \cdot T_2 = T_{11}T_{21} + T_{12}T_{22} + T_{13}T_{23} = \sin\delta$. The parameter δ is useful in measuring how much the calibration matrix deviates from a true perspective projection matrix. It is a convenient way of lumping into a single parameter this deviation.

A brief summary of the procedure described in this section is shown in Fig. 3.9.

3.6 A COMPARISON OF THE CALIBRATION METHODS

The ideal pinhole method of Section 3.2 requires that the calibration points be non-coplanar in the world frame. As discussed there, with a coplanar set of points it may not be possible to obtain a solution at all, especially if the procedure of Section 3.2.2 is used for computation.

The procedure proposed by Tsai, described in Section 3.3, has the advantage of allowing for coplanar calibration points in the world frame. Using coplanar calibration points simplifies the physical set up required for the calibration process since fabricating a coplanar set of points is much easier than a set of non-coplanar points, especially so because the world coordinates of these points must be known accurately.

The main reason Tsai's method does not require that co-planarity constraint be satisfied by the calibration points is due to the additional constraints that are invoked, such as the orthonormality of the rotation matrix. Therefore the perspective equations need be solved for only a smaller number of unknowns, the rest of the unknowns are then computed using the orthonormality constraint. But note that in order to use this procedure, in accordance with the first step in Fig. 3.7, we must first convert the the discrete

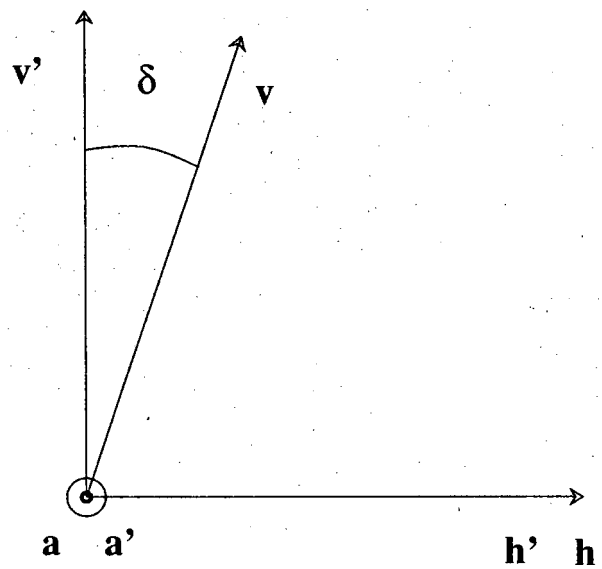


Figure 3.8: The skew angle δ.

1: Compute $\left[\vec{C}\cdot\hat{a}\right]^2$ from $T_3 \cdot T_3$.

2: Compute i_0 and j_0 from $T_1 \cdot T_3$ and $T_2 \cdot T_3$ respectively.

3: Compute $\frac{f}{\Delta u}^2$ and $\frac{f}{\Delta v}^2$ from the $T_1 \cdot T_1$ and $T_2 \cdot T_2$ respectively.

4: Assuming the sign of $-\vec{C}\cdot\hat{a}$, $\frac{f}{\Delta u}$ and $\frac{f}{\Delta v}$ positive compute $\hat{h}$, $\hat{v}$ and $\hat{a}$ from T_1, T_2 and T_3 respectively.

5: If $\hat{h}$, $\hat{v}$ and $\hat{a}$ do not form a right hand base then reverse the sign of the three components of $\hat{v}$.

6: Compute $\sin\delta$ from $T_1 \cdot T_2$.

Figure 3.9: The steps involved in obtaining the camera characteristics from the entries in the calibration matrix. Observe that the procedure only involves the solution of linear equations.

coordinates of an image point corresponding to a scene calibration point into what were referred to as the distorted image coordinates -- these are the actual analog coordinates of an image point, taking into account the lens distortion, etc. This conversion requires that we know *a priori* the coordinates of the image center, the sampling intervals in the image plane, etc. Although the sampling-interval information can be gleaned from the product information supplied by the manufacturer, one still has to figure out the location of the image center before the method can be used. Also needed is what was called the uncertainty image scale factor.

In [9], Tsai did show that by using some calibration points that are non-coplanar with the points used in the procedure described in Section 3.3, it is possible to compute the uncertainty image scale factor. This modification to the calibration procedure also had the advantage that the image center could now be anywhere in the image plane without affecting the accuracy of calibration. We found, however, that this claim is true only if the objects points in the scene stay close to the plane used originally for calibration. In a later paper [10], Tsai recognized this problem and proposed another technique to determine the remaining uncalibrated parameters, such as the coordinates of the image center and the uncertainty image scale factor.

The last column in Table 3.1 summarizes the various features of the Tsai method as discussed in Section 3.3. The topmost entry in the column shows the parameters that must be known *a priori* before the calibration method can be used. The other entries are self-explanatory.

We chose not to use the Tsai method for calibrating the cameras on the mobile robot because of the need to know the location of the intersection of the image plane and the optic axis (this intersection defines the center of the image plane) and because the advantage of the coplanarity of the calibration points was really not an issue in our work. Non-coplanar points whose world coordinates are precisely known are easy to come by in scenes recorded by the cameras on the mobile robot.

In the case of the two-plane method discussed in Section 3.4, we encountered two major drawbacks. One disadvantage that was mentioned before is that while the method gives us a procedure for predicting lines of sight corresponding to the pixels in an image, it does not tell us how to render an image from a 3D model of the world. In other words, the two-plane method does not tell us which pixel corresponds to a given object point in a scene, but only which line-of-sight corresponds to a given pixel in the image. The second drawback has to do with the locations of the two calibration planes in relationship to the camera. We found that for best results with this method, the two calibration planes should be located in such a manner that all the object points of interest will lie between the two planes. For mobile robotics that presents a problem because of the large depths associated with the scenes. Large depths mean that the farther calibration plane must be at a large distance from the camera. Even if we could arrange for a calibration plane at a

Table 3.1: Comparison of the characteristics of three camera calibration methods.

Method	Perspective Transform	Two Planes	Tsai
Parameters Uncalibrated	None	None	$C_x, C_y,$ s_x, d_x, d_y
Coplanar Calibration Points	No	No	Yes
Accounts for distortion	No	Yes (Linear splines)	Radial only
Involves Nonlinear Search	Yes	No	Yes

large distance, say at a distance of 20 meters from the camera, there would still be another difficulty to resolve. For best results, it is desirable that the scene points used for calibration be uniformly distributed over each of the calibration planes. The extent of each calibration plane covered by such a distribution of points depends on the view angle of the camera. The cameras we use on the mobile robot have a field of view of 26°. At a distance of 20 meters, this angle results in a rather large area and arranging for a uniformly distributed set of points over this area with accurately known world coordinates seemed too difficult a task.

For our mobile robotics work we therefore chose to use the method which is described in Section 3.2 and which is based on the assumption of an ideal pinhole for the camera lens. Experimentation showed that the lens distortion was not a problem; the solid-state cameras usually tend to use high-quality lenses.

3.7 CALIBRATION PROCEDURE USED FOR NAVIGATION EXPERIMENTS

As mentioned in the preceding section, we use the method of Section 3.2 for calibrating the cameras on the mobile robot. To supply this method with non-coplanar scene points whose world coordinates are known, we initially used a flat board with a pattern of large circular black dots, the board would then be placed at various distances from the robot. At each location of the board, the circular dots were first detected and their centroids found, these centroids were then used as scene points for calibration. This procedure was completely automated, all the operator had to do was to key in the coordinates of the corners of the board at each of its positions. The output of the calibration program was the transformation matrix T of Eq. (3.11).

A disadvantage of this procedure was that the board with the black dots had to be positioned at precisely known locations in relation to the position of the robot. To the extent that could not be done, errors tended to corrupt the calibration procedure. Of course, instead of moving the board, one could move the robot in relation to the board (in fact, that's what we did in most of our experiments). However, moving the robot did not alleviate the problem due to the errors in the odometry of the robot; these errors are a function of the slippage between the wheels and the floor. Another disadvantage of this procedure was the rather poor accuracy with which the image processing routines tended to locate the centroids of the black dots; usually the accuracy was not better than two or three pixels.

For these reasons, we have now switched to an interactive approach in which the robot is placed at a fixed position. The image recorded by the camera is displayed on a SUN window and the operator, who has available to him/her precise coordinates of many scene points such as the corners of walls, door frames, panels, etc., uses mouse clicks to

select a set of non-coplanar scene points from the image and to enter the world coordinates of the selected points. Actually, the operator selects more points than needed for calibration, the extra points are then used for testing the accuracy of the calibration matrix by predicting the image coordinates of these extra points and comparing the predictions with the actual image coordinates.

For this interactive calibration procedure, we choose a point on the floor and designate it as the origin of the world frame. All the coordinate measurements to the candidate scene points such as the corners of walls, door frames, panels, etc., are then made with respect to this origin and recorded. The robot is subsequently placed such that the center of its base is right over the origin. For calibration purposes, with the robot located as described, the axes of the world frame are defined such that the Z axis is vertical with the positive direction being the up direction; the positive direction of the Y axis coincides with the forward direction of motion of the robot and the X axis is chosen so the resulting frame is right handed. The world frame defined in this manner for the purpose of calibration becomes eventually the robot frame during navigation; in other words, it becomes the frame that always stays with the robot. All the lines of sight computed via the calibration matrix are with respect to this frame.

To help the operator with the task of entering the world coordinates of the selected scene points, a SUNVIEW based program called *suncal* was written. This program allows the use of a mouse button to bring up a text window for the entry or the erasure of the selected scene points. By selecting appropriate menu items, it is also possible to read the coordinates of the already selected points from a unix file. If the *show* mode is selected from the menu, the world and the image coordinates of a point are displayed and the corresponding point highlighted in the image to check for possible mistakes. Fig. 3.10 shows a typical image used during this calibration procedure and some of the pop-up menus, however they might have become invisible through duplication.

3.8 TEST RESULTS

Table 3.2 shows a set of 14 scene points used for the calibration of the two cameras on the robot. The world coordinates of the points are shown in the left half of the table, and the corresponding image coordinates shown in the right half for each of the two cameras. These points were selected from the hallway pictures of Fig. 3.11. In Table 3.2, the image coordinates under 'Left' were extracted from the left camera image shown at the top in Fig. 3.11, and those under 'Right' from the right camera image shown at the bottom in Fig. 3.11.

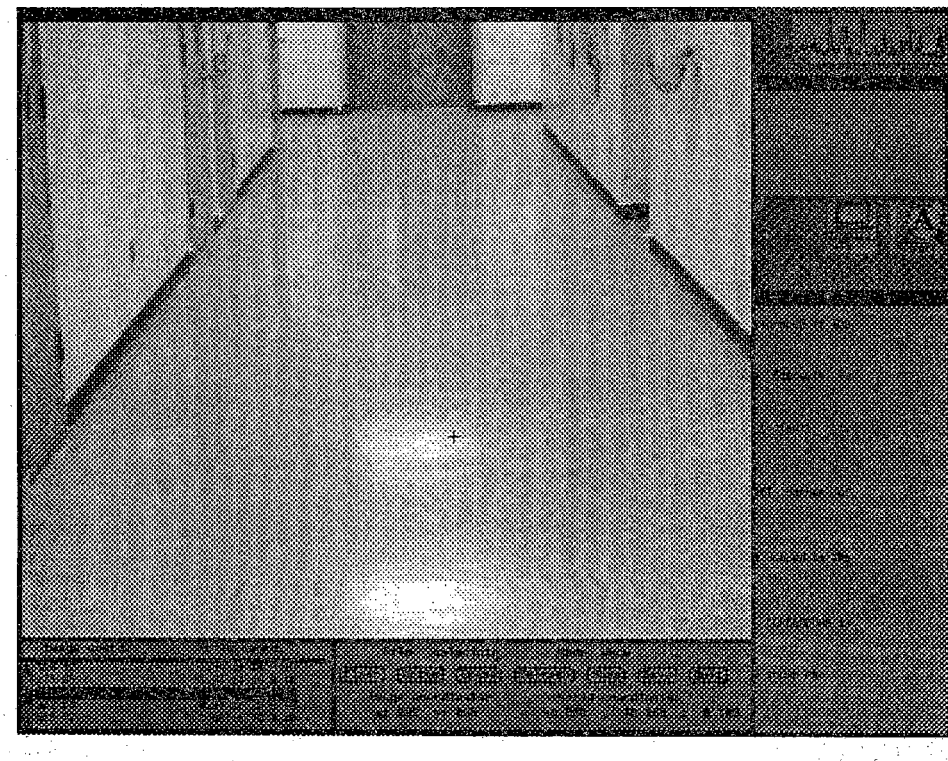


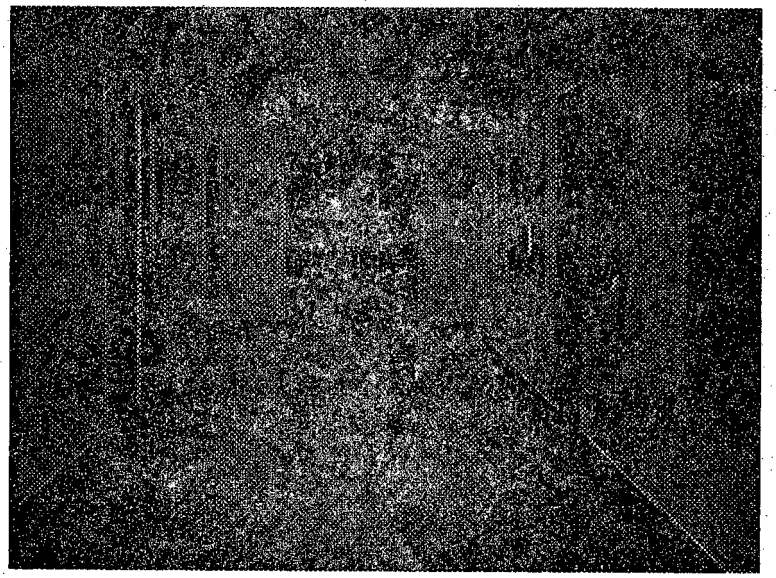
Figure 3.10: The suncal tool in execution.

Table 3.2: Coordinates of the calibration points used.

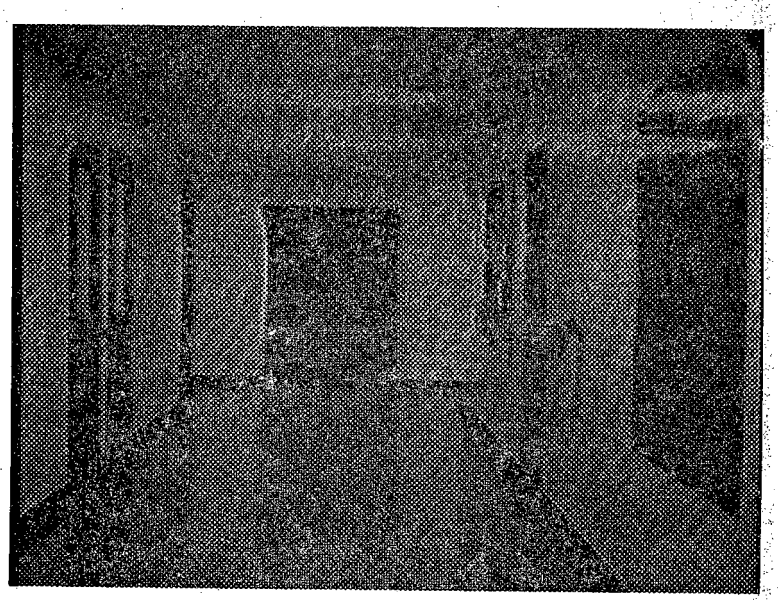
Calibration Points						
World Coord. (m)			Image Coord.			
			Left		Right	
X	Y	Z	U	V	U	V
-1.34	8.69	2.19	62	55	38	81
-1.34	8.69	0.10	63	339	37	367
-1.34	9.71	2.19	80	67	57	94
-1.34	9.71	0.10	80	321	57	347
-1.34	14.50	0.10	126	273	108	300
-0.81	18.26	2.19	172	115	158	142
-0.81	18.26	0.10	172	250	157	279
0.97	18.26	2.19	265	114	251	143
0.97	18.26	0.10	264	251	252	280
1.48	5.96	0.67	430	304	397	334
1.48	8.79	2.19	366	59	343	87
1.48	8.79	0.10	365	338	340	368
1.48	9.97	2.19	350	71	329	100
1.48	9.97	0.10	348	317	326	349

Table 3.3: Coordinates of the test points used.

Test Points						
World Coord. (m)			Image Coord.			
			Left		Right	
X	Y	Z	U	V	U	V
-1.34	10.08	2.19	84	70	61	97
-1.34	10.08	0.10	85	316	61	343
-1.34	11.10	2.19	97	79	77	107
-1.34	11.10	0.10	98	302	77	328
-1.34	14.50	2.19	125	100	108	129
1.48	15.79	0.10	304	264	288	294
1.48	10.65	2.14	342	83	322	112
1.48	4.20	2.08	-	-	470	3
1.48	4.20	0.67	-	-	467	391



(a) Left Image



(b) Right Image

Figure 3.11: Stereo hallway images.

The procedure described in Section 3.2 was used for the computation of the calibration matrices T for each of the two cameras. For each camera, first the system of equations in (3.24) was solved for an optimal U in the least squares sense; this solution was then used as an initial guess for a nonlinear optimization procedure that solves equations (3.21). The residuals at each step during the computation of the calibration matrix from the data of Table 3.2 are shown in Table 3.4. The first column is the residual $(A\,U - B)^T\,(A\,U - B)$ corresponding to the linear least squares solution (LSS) to the linear system (LS) of equation in (3.24). The second column was obtained using this solution for computing the residual of the non-linear system (NLS) of equations (3.21) for $l = 1,...,N$. The third column represents the residual for the nonlinear equations of (3.21) after an optimum solution is found via nonlinear optimization (NLSS). The calibration matrices, T, obtained for the two cameras are shown in Fig. 3.12.

After obtaining the calibration matrices, the physical parameters for each of the cameras are computed following the procedure described in Section 3.5. By comparison with the parameters used in setting up the cameras, some of these computed physical parameters can then be used to verify the calibration procedure. The values found for the physical parameters from the calibration matrices of Fig. 3.12 are shown in Table 3.5. The coordinates of the lens center may be computed either by using Eq. (3.14) -- this is labeled as the 'Direct' method in Table 3.5 -- or by using the Ganapathy procedure of Section 3.5 which is based on the Eqs. (3.52d) and (3.52h). The results of both computations are shown in the table for comparison. As was mentioned before, the angle δ is a measure of the appropriateness of the lens model used for calibration. A large value of δ would indicate that the results might not be meaningful. The column labels 'Left' and 'Right' refer to the left and the right cameras on the robot.

The accuracy of the calibration was checked using three different tests. The first test was to compute the image coordinates of a number of scene points given their world coordinates. The results shown in Table 3.6 correspond to the test points of Table 3.3. Note that an error smaller than 0.5 pixels is equivalent to no error at all for all practical purposes.

For the second test, we computed the line-of-sight corresponding to each test point given its image coordinates. Results for the test points of Table 3.3 are presented in Table 3.7. Clearly, before a line-of-sight corresponding to a given image pixel can be computed, we must get a fix on the camera lens center, since all lines of sight emanate from the lens center. As with the results in Table 3.5, we used two separate methods for computing the coordinates of the lens center for each of the cameras: the direct method using Eq. (3.14) and the Ganapathy method using Eqs. (3.52d) and (3.52h) were employed. Table 3.7 presents the minimum, the maximum, and the average errors over all the test point of Table 3.3 using lens center locations derived from the two separate methods. To calculate the error in the line-of-sight, we first compute the angle of the ray associated

Table 3.4: Residuals for different solutions of the calibration matrix.

Sum of Squares of Residuals			
Image	LSS in LS	LSS in NLS	NLSS in NLS
Left	2542.9	7.6	2.5
Right	1329.7	5.5	2.5

$$\begin{bmatrix} 3.503841e{+}03 & 7.993313e{+}02 & -2.075188e{+}01 & -1.910881e{+}02 \\ 2.820852e{+}01 & 6.104280e{+}02 & -4.404421e{+}03 & 6.153614e{+}03 \\ 8.101512e{-}02 & 3.626108e{+}00 & -1.363650e{-}01 & 1.0 \end{bmatrix}$$

(Left)

$$\begin{bmatrix} 2.970296e{+}03 & 6.625147e{+}02 & 3.098967e{+}00 & -7.379562e{+}02 \\ 4.269193e{+}01 & 6.026127e{+}02 & -3.751580e{+}03 & 5.265578e{+}03 \\ 8.539027e{-}03 & 3.055924e{+}00 & -9.756639e{-}02 & 1.0 \end{bmatrix}$$

(Right)

Figure 3.12: Left and right calibration matrices.

Table 3.5: Physical parameters obtained from the calibration matrices of fig. 3.11.

Camera physical parameters			
Parameter	Method	Left	Right
θ pan	Ganapathy	-1.28°	-0.16°
ϕ tilt	Ganapathy	92.15°	91.8°
ψ swing	Ganapathy	179.80 °	179.5°
C_x	Ganapathy	-11.40 cm.	7.82 cm.
C_x	Direct	-11.42 cm.	8.19 cm.
C_y	Ganapathy	0.16 cm.	-5.61 cm.
C_y	Direct	0.17 cm.	-5.60 cm.
C_z	Ganapathy	1.36 m.	1.36 m.
C_z	Direct	1.36 m.	1.36 m.
K_u	Ganapathy	960.2 pix/m.	970.9 pix/m.
K_v	Ganapathy	1206.3 pix/m.	1220.2 pix/m.
i_0	Ganapathy	241.8 pixels	219.2 pixels
j_0	Ganapathy	213.8 pixels	236.2 pixels
δ	Ganapathy	-0.010°	0.157°

with an image point corresponding to a test scene point, as given by Eq. (3.20). We then compute angle of the ray to the test scene point as given by the world coordinates of the test point and the calculated location of the camera lens center. The difference between the two angles is the error in the line-of-sight.

For the third test, we used stereo triangulation on the two lines of sight from the left and the right cameras to calculate the 3D coordinates of a scene test point. A comparison of the 3D coordinates thus obtained and the actual 3D coordinates of the test point yielded a measure of the error. The results are shown in Table 3.8. The two rows refer to the different procedures used for computing the location of the camera lens centers. The camera baseline used in these results was 19.6 cm.

Table 3.6: Radius of error for the points in table 3.3.

Radius of error (in pixels)			
Image	min	max	mean
Left	0.0	2.1	0.6
Right	0.0	3.0	1.0

Table 3.7: Line of sight angular error for the points in table 3.3.

Line of sight errors (degrees)				
Image	Lens center	min	max	mean
Left	Direct	0.0139	0.1309	0.0495
Left	Ganapathy	0.0130	0.1318	0.0501
Right	Direct	0.0393	0.1471	0.0898
Right	Ganapathy	0.0306	0.1858	0.0973

Table 3.8: Stereo triangulation error for the points in table 3.3.

Stereo radial errors						
Lens center	Absolute error			Relative error		
	min	max	mean	min	max	mean
Direct	0.097 m.	1.086 m.	0.471 m.	0.9 %	6.9 %	3.8 %
Ganapathy	0.070 m.	1.387 m.	0.630 m.	0.6 %	8.7 %	5.0 %

REFERENCES

[1] R. O. Duda and P. E. Hart, *Pattern Recognition and Scene Analysis,* Wiley, New York, 1973.

[2] D. H. Ballard and C. M. Brown, *Computer Vision,* Prentice-Hall, 1982.

[3] *Manual of Photogrammetry,* American Society of Photogrammetry, 1980.

[4] Y. Yakimovsky and R. Cunningham, "A system for extracting three dimensional measurements from a stereo pair of TV cameras", *Computer Graphics Image Processing,* vol. 7, pp. 195-210, 1978.

[5] H. Moravec, *Robot Rover Visual Navigation,* UMI Research Press, 1981.

[6] S. Ganapathy, "Decomposition of transformation matrices for robot vision", *Proc. Int. Conf. Robotics and Automation,* pp. 130-139, 1984.

[7] K. W. Wong, "Mathematical formulation and digital analysis in close range photogrammetry", *Photogrammetric Eng. Remote Sensing,* vol. 41, pp. 1355-1373, 1975.

[8] R. Y. Tsai, "A versatile camera calibration technique for high accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses", *Proc. IEEE Conference on Computer Vision and Pattern Recognition,* pp. 363-374, 1986.

[9] R. Y. Tsai, "A versatile camera calibration technique for high accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses", *IEEE Journal of Robotics and Automation,* vol. RA-3 NO. 4, pp. 323-344, 1987.

[10] R. K. Lenz and R. Y. Tsai, "Techniques for calibration of the scale factor and image center for high accuracy 3D machine vision metrology", *Pattern Analysis and Machine Intelligence,* vol. 10, pp .713-720, 1988

[11] H. A. Martins, J. R. Birk, and R. B. Kelley, "Camera models based on data from two calibration planes", *Computer Graphics Image Processing,* vol. 17, pp. 173-180, 1981.

[12] A. Izaguirre, P. Pu, and J. Summers, "A new development in camera calibration: Calibrating a pair of mobile cameras", *Proc. Int. Conf. Robotics and Automation,* pp. 74-79, 1985.

[13] K. D. Gremban, C. E. Thorpe, and T. Kanade, "Geometric Camera Calibration using systems of linear equations", *Proc. Int. Conf. Robotics and Automation,* pp. 562-567, 1988.

[14] A. C. Kak, "Depth perception for robots", *Handbook of Industrial Robotics*, pp. 272-319, John Wiley, New York, 1985.

[15] K. M. Andress and A. C. Kak, "Evidence Accumulation and Flow of Control in a Hierarchical Spatial Reasoning System", *AI magazine*, vol. 9, no. 2, pp. 75-95, 1988.

CHAPTER 4

FEATURE EXTRACTION

First of all, a couple of words on why vision feedback is needed for our work in mobile robot navigation. If the odometry on the robot were perfect, the robot would be able to navigate in the blind from its start position to the destination just as an airplane can be flown in the blind on instruments. However -- and fortunately for researchers such as ourselves -- the odometry on most mobile robots is very poor. To give the reader an idea of the poor quality of odometry on our robot, in many instances a commanded turn of 45^o introduces an orientation uncertainty of 2^o and a commanded straight-line motion has associated with it about 10% uncertainty in the location of the robot at the end of the motion. What's worse, due to uneven weight distribution in the base of the robot where a heavy battery is housed, a command to travel straight in a certain direction usually results in motion along a line that could be up to 15^o off from the commanded direction. It is not possible to construct a usable model of this uncertainty as the uncertainties depended strongly on factors such as the starting orientation of the robot, how recently the floors have been waxed, etc.

To compensate for poor odometry, the robot needs some other sensory feedback; in our mobile robot, vision from one or both cameras is used. Due to the time it takes to process a frame of vision data, this compensation in our current system takes place in a discrete mode, meaning that the robot travels in the blind (except for the use of ultrasonic sensors for collision avoidance) and every so often (currently this distance is 6 meters), the robot stops, analyzes the images from one or both the cameras, and figures out where exactly it is in the world. This periodic exercise by the mobile robot is called *self-location*. When both cameras are used for self-location, the images from the two cameras are fused by binocular stereopsis and, through this fusion, a distance map to some of the "prominent" features of the scene created. If the geometry of the prominent features matches that of what the robot expected to see -- within of course the scope of uncertainties in the odometry -- the distance map generated is then used to figure out the precise location of the robot in relation to the world model. More on this in Chapter 5.

When only a single camera is used for self-location, the image from the camera is compared with an expectation map that is rendered from a CAD model of the hallways.

The PSEIKI reasoning system is used for this comparison. This method is described in Chapter 6.

In this chapter, we will only be concerned with how to extract vertical straight edges from an image so that they can be used in the first of the two methods described above -- the method of binocular stereopsis discussed in Chapter 5.

The approach we discuss consists of the following steps: First the local edges in the images are detected and thinned, this step is called edge detection. Then the local edges are grouped into lines and the parameters of the lines, such as their locations and orientations, estimated; this process is called line detection. The estimated line parameters are improved by using a Hough transform based algorithm.

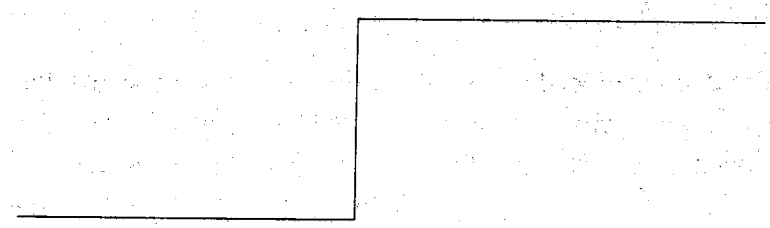
4.1 LOCAL EDGE DETECTION

An ideal edge in one dimension may be viewed as a step change in intensity. In real images the change of intensity is likely to occur over a finite length and also to be corrupted by noise (Fig. 4.1). Since edges are high-spatial-frequency phenomena, edge finders must of necessity be sensitive to high frequency noise. While the detection of ideal edges uncorrupted by noise would be simple, in practice a compromise must be achieved between maximizing the detection of the desired image edges and minimizing the detection of undesired noise edges.

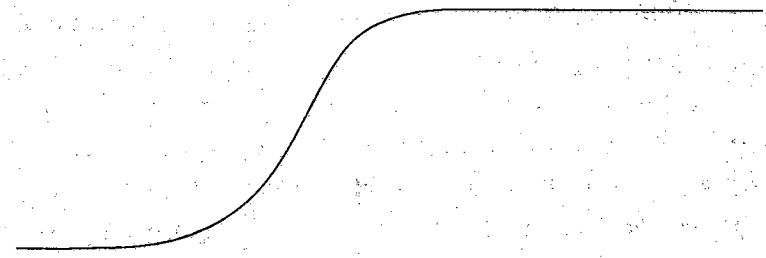
To perform the task of detecting local edges a wide variety of *edge operators* have been developed. An edge operator is a mathematical function of small spatial extent designed to detect the presence of a local edge in an image via convolutional processing. Edge operators may be classified into three main classes:

- Operators that approximate the mathematical gradient or Laplacian. We will refer to such operators by the name *difference operators*.
- Template matching operators that use multiple templates at different orientations.
- Operators that fit local intensities with parametric edge models.

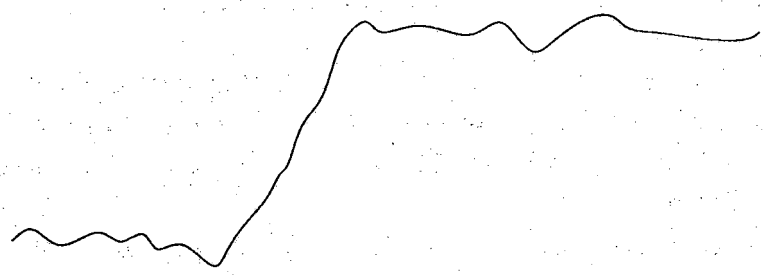
In the next subsection, we will briefly review some operators in each of these classes. For a more complete discussion, the reader is referred to [1], [2], [3].



(a)



(b)



(c)

Figure 4.1: Edge cross sections: gray level changes across the edge (a) Perfect step edge. (b) Noise free blurred edge. (c) Noisy blurred edge.

4.1.1 Difference Operators

The most common edge-detection operators in this category are discrete approximations to the mathematical gradient and Laplacian.

a. The Gradient

The gradient of a function f indicates the direction in which the rate of change of f is the largest and the magnitude of this rate of change. For a two-dimensional function such as an image the direction and the magnitude of the gradient are given by

$$\theta = \tan^{-1}\left[\frac{\frac{\partial f}{\partial y}}{\frac{\partial f}{\partial x}}\right] \tag{4.1}$$

$$|\,grad\, f\,| = \sqrt{\left[\frac{\partial f}{\partial x}\right]^2 + \left[\frac{\partial f}{\partial x}\right]^2} \tag{4.2}$$

where x and y are two perpendicular directions and the angle θ is measured from the x-axis. For a digital picture, the continuous derivatives are approximated as differences. Therefore, we may write

$$\theta = \tan^{-1}\left[\frac{\Delta_2}{\Delta_1}\right] \tag{4.3}$$

$$|\,grad\, f\,| = \sqrt{\Delta_1{}^2 + \Delta_2{}^2} \tag{4.4}$$

where Δ_1 and Δ_2 are the finite differences approximations to the derivatives along two perpendicular directions. One possibility could be

$$\begin{aligned} \Delta_1 &\equiv f(x, y) - f(x-1, y) \\ \Delta_2 &\equiv f(x, y) - f(x, y-1) \end{aligned} \tag{4.5}$$

From the standpoint of implementation, Δ_1 and Δ_2 can be construed to be convolutional operators, meaning that the value of Δ_1 and Δ_2 at each pixel may be computed by convolving the digital image with the patterns:

$$\begin{bmatrix} -1 & 1 \end{bmatrix} \qquad \text{and} \qquad \begin{bmatrix} 1 \\ -1 \end{bmatrix} \tag{4.6}$$

The approximations to the continuous derivatives, as represented by Eqs. (4.5), suffer from the fact that if we had to assign locations to the points at which the x- and the y-components of the rates of change of the function f are computed, these locations would not be coincident with the point (x,y); what's worse, the location for the x-component is

not the same as that for the y-component. To explain, from the constructions of the right-hand-sides in Eqs. (4.5), one may say that Δ_1 is centered at $(x - \frac{1}{2}, y)$, whereas Δ_2 is centered at $(x, y - \frac{1}{2})$, making questionable any attempt at combining the two components via Eqs. (4.3) and (4.4). Such problems do not exist with other more symmetrical approximations to the derivatives as represented by the operators displayed in (b) and (c), the latter called the Roberts operator, of Fig. 4.2, the (a) operator shown there is the same as in Eq. (4.6) above.

The adverse effect of noise can be reduced by locally averaging the picture before the application of the operators shown in Fig. 4.2. From a computational standpoint, one can equivalently compute the differences of local averages with the application of operators such as those shown in Fig. 4.3. The Prewitt and the Sobel operators shown in this figure, especially the latter one, are probably the most commonly used operators for edge detection in digital image processing. Note that in both the operators, when we compute x-component of the rate of change, we also do averaging along the y-direction; and when we compute the y-component of the rate of change, we do averaging in the x-direction.

b. The Laplacian

The Laplacian is an orientation invariant derivative operator given by

$$\nabla^2 f(x, y) \equiv \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} . \tag{4.7}$$

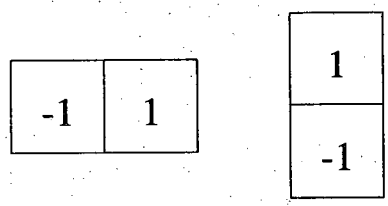
One discrete approximation to the Laplacian is

$$L(x, y) \equiv f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) + 4f(x, y) \tag{4.8}$$

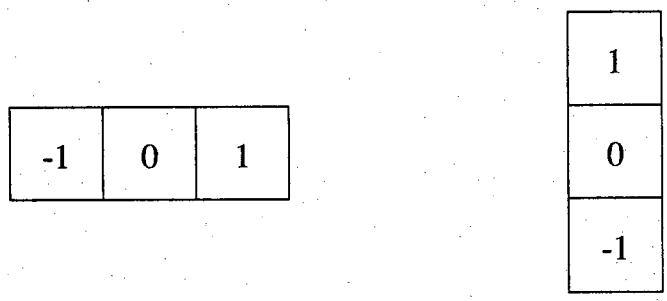
which may be implemented by carrying out a digital convolution of f with

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} . \tag{4.9}$$

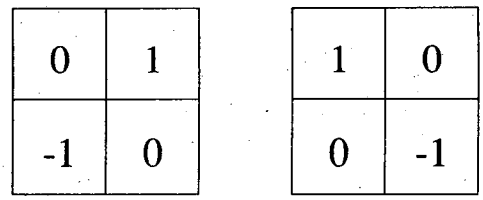
The Laplacian, being a second-difference operator, has a "double spike" response to a perfect step edge, as illustrated by Fig. 4.4a for the one-dimensional case. For a perfect ramp, the output spikes are located at the "shoulders" at the top and the bottom of the ramp, as shown in Fig. 4.4b. For a more realistic edge, as in Fig. 4.4c, the output exhibits a zero-crossing at a point half way between the high and the low associated with the edge. For this reason, Laplacian processing must be followed by a zero-crossing detector, since the points where the zero-crossings do occur are presumed to be the locations of edges.



(a)

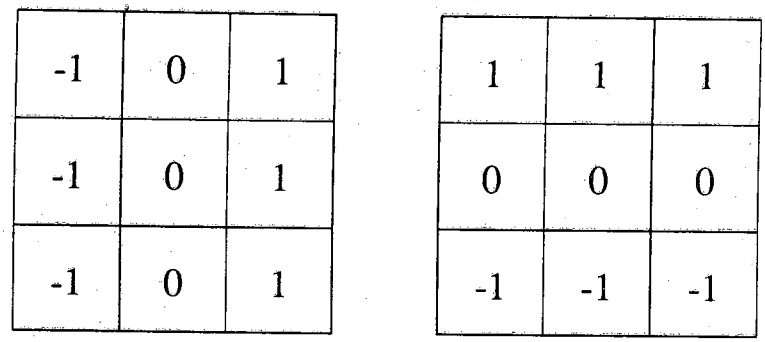


(b)



(c)

Figure 4.2: Gradient operators.



-1	0	1
-1	0	1
-1	0	1

1	1	1
0	0	0
-1	-1	-1

(a)

-1	0	1
-2	0	2
-1	0	1

1	2	1
0	0	0
-1	-2	-1

(b)

Figure 4.3: Prewitt (a) and Sobel (b) operators.

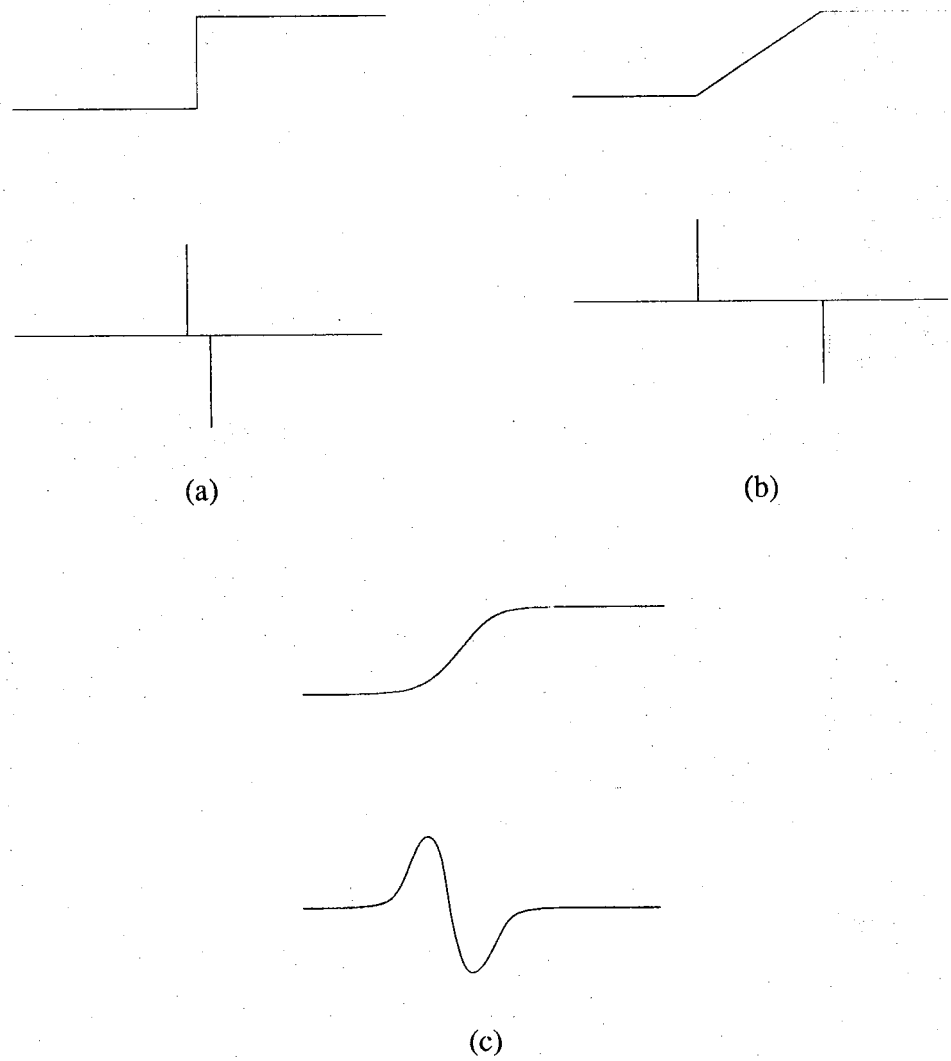


Figure 4.4: Digital Laplacian response to different types of edges. (a) perfect step, (b) perfect ramp, (c) blurred edge.

The Laplacian has three disadvantages as an edge detector: 1) It does not provide edge directional information explicitly, although it must be said that the directional information can be inferred from the orientations of the zero-crossing contours. 2) Since the second derivatives are involved, we get a double enhancement of noise in the image. And, 3) digital Laplacian responds more strongly to corners, lines, line ends, and isolated points than it does to edges; this point has been discussed in [1]. To reduce the effects of noise and also to somewhat reduce the sensitivity of the Laplacian operator to corners, lines and line ends, it is common to first smooth an image with a Gaussian function described by

$$G_\sigma(x, y) = \sigma^2 e^{-(x^2+y^2)/2\sigma^2} \tag{4.10}$$

In practice, both the Laplacian and the smoothing can be packaged into a single operator that is a convolution of the two operators. As a continuous function, the single operator that embodies both smoothing and the second-derivative operation is given by

$$\nabla^2 G_\sigma * f(x, y) = \left[\frac{r^2 - 2\sigma^2}{\sigma^4}\right] \exp\left[\frac{-r^2}{2\sigma^2}\right] \tag{4.11}$$

where

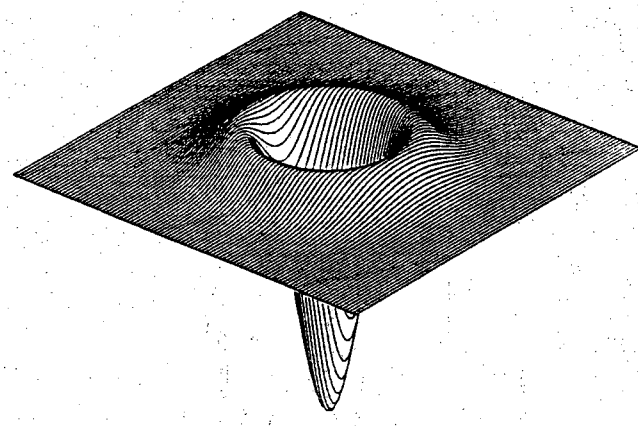
$$r = \sqrt{x^2 + y^2} \tag{4.12}$$

This is a rotationally symmetric function with one free parameter, σ, which determines the spatial size of the function -- the spatial size of the function controls the amount of smoothing. The choice of σ is a compromise between the resolution we wish to achieve for edge detection and the amount of noise we need to filter out. Marr and Hildreth [4] have suggested the use of different values of σ to correspond to the different bandpass channels of the human visual system. Further discussion on this approach to edge detection, also called the Laplacian-of-Gaussian approach, can be found in [4] and [5]. In Fig. 4.5, we have shown a plot of the Laplacian-of-Gaussian operator of Eq. (4.11).

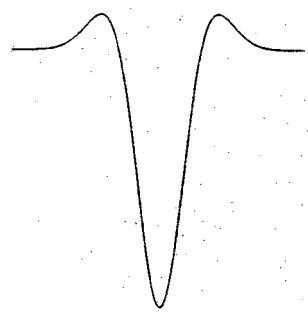
4.1.2 Template Matching

Another approach to detecting edges consists of convolving an image with various templates, each corresponding to an ideal edge at a particular orientation. At each pixel, we then take for edge orientation that value which corresponds to the template yielding the largest value, the magnitude of the edge strength being the local value of the convolution with the template. Frequently in the literature, such templates are also referred to as edge detection masks.

An important parameter to select in this approach to edge detection is the size to use for the masks; larger masks offer greater immunity to noise but reduced resolution. That



(a)



(b)

Figure 4.5: Laplacian of a Gaussian. (a) 3D profile, (b) intersection with a plane containing the z axis.

edge detection resolution is diminished with larger masks is owing to the fact that the convolved output is non-zero over an extent whose size depends directly on the size of the mask. If the convolved output is simply thresholded, the result is usually a thick detected edge, the thickness being proportional to the size of the mask, as illustrated in Fig. 4.6. Usually, it is possible to improve the resolution by thinning the output of the edge detector by a number of techniques such as the one described by Eberlein [6].

Nevatia and Babu [7] have used the templates shown in Fig. 4.7 in a system used to extract linear features suitable for detecting roads and airport runways from aerial photographs. A method for thinning the output of the edge detector and another to link broken segments are also presented.

4.1.3 Edge Fitting

Another approach to edge detection is to have parametric models of ideal edges and to determine how close these models fit the neighborhood of a given image point. One of the best known procedures based on this parametric model approach is due to Hueckel [8], [9].

A simplified model of a general edge that is used to derive the Hueckel's operator is shown in Fig. 4.8. It represents an edge at a distance r from the center of a circular region that is being analyzed for the presence or absence of an edge; the edge is assumed to be at an orientation of angle θ and, further, it is assumed that the edge separates the circular region into two areas of uniform brightness, of values b and $b+h$. We want to compute the parameters b, h, r and θ of the ideal step function that best matches the given image region.

We will let N^2 denote the sum of the pixel-by-pixel squared differences between the actual gray level distribution in a circular region R of an image and an ideal step edge, as depicted in Fig. 4.8, in the same region:

$$N^2 = \sum_{\mathbf{x} \in \mathbf{R}} (A(\mathbf{x}) - S(\mathbf{x}, \xi))^2 \tag{4.13}$$

where $A(\mathbf{x})$ is the gray value of the image at point $\mathbf{x}$ and $S(\mathbf{x}, \xi)$ is the gray value of the ideal step corresponding to a given set of parameters ξ, also at $\mathbf{x}$. The vector of parameters, ξ, has to be chosen such that N^2 is minimized. An edge is declared present in R if N^2 is sufficiently small and the step height, h, is sufficiently large.

Hueckel proposed a procedure for minimizing N^2 in which both $A(\mathbf{x})$ and $S(\mathbf{x}, \xi)$ are expressed in terms of an orthogonal set of functions that are particularly appropriate for capturing different types of gray level transitions in R. In terms of the coefficients of such orthogonal expansions, the error measure N^2 can be expressed as

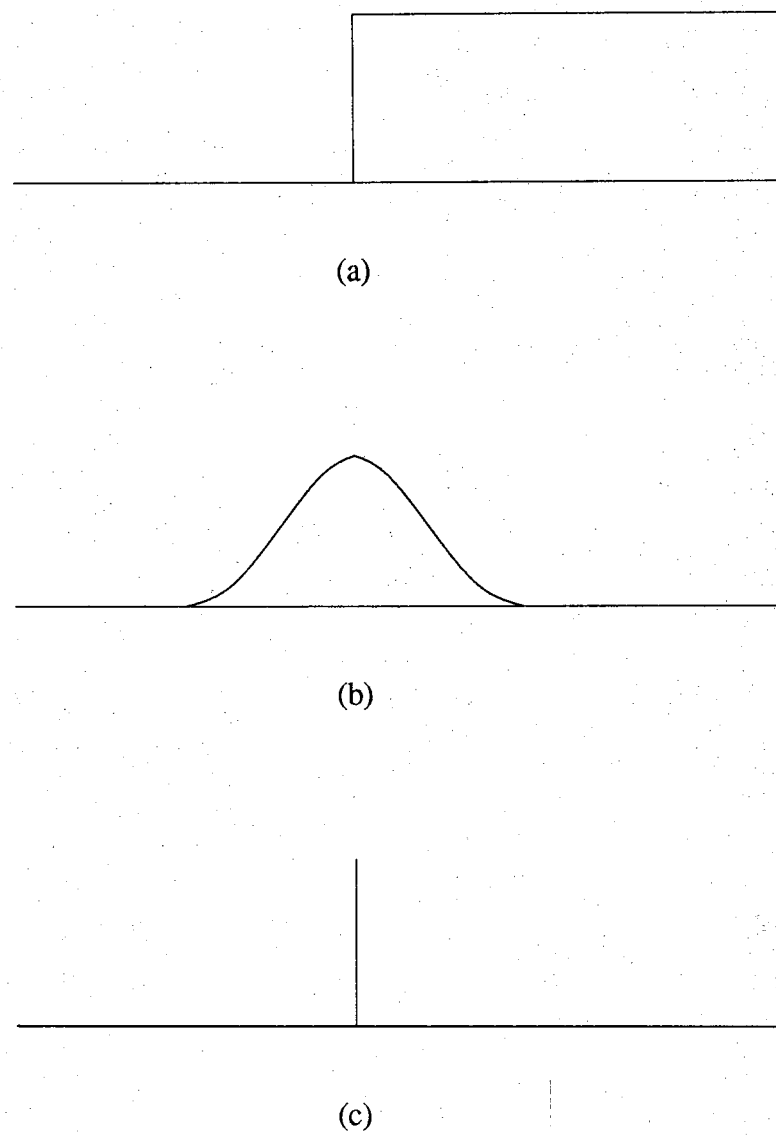


Figure 4.6: Edge detector output profile. (a) perfect step edge, (b) output of a "wide" mask, (c) desired output after thinning.

−100	−100	0	100	100
−100	−100	0	100	100
−100	−100	0	100	100
−100	−100	0	100	100
−100	−100	0	100	100

(*a*) 0°

−100	32	100	100	100
−100	−78	92	100	100
−100	−100	0	100	100
−100	−100	−92	78	100
−100	−100	−100	−32	100

(*b*) 30°

100	100	100	100	100
−32	78	100	100	100
−100	−92	0	92	100
−100	−100	−100	−78	32
−100	−100	−100	−100	100

(*c*) 60°

100	100	100	100	100
100	100	100	100	100
0	0	0	0	0
−100	−100	−100	−100	−100
−100	−100	−100	−100	−100

(*d*) 90°

−100	100	100	100	100
−100	100	100	78	−32
−100	92	0	−92	−100
32	−78	−100	−100	−100
−100	−100	−100	−100	−100

(*e*) 120°

100	100	100	32	−100
100	100	92	−78	−100
100	100	0	−100	−100
100	−78	−92	−100	−100
100	32	−100	−100	−100

(*f*) 150°

Figure 4.7: Edge masks in six directions: (a) 0°, (b) 30°, (c) 60°, (d) 90°, (e) 120°, (f) 150°.

$$N'^2 = \sum_{i=0}^{8} (a_i - s_i(xi))^2 \tag{4.14}$$

where a_i and s_i are given by

$$a_i = \sum_{\mathbf{x} \in \mathbf{R}} H_i(\mathbf{x}) \cdot A(\mathbf{x}) \tag{4.15}$$

$$s_i = \sum_{\mathbf{x} \in \mathbf{R}} H_i(\mathbf{x}) \cdot S(\mathbf{x}) \tag{4.16}$$

the functions H_i being the orthogonal basis functions. Fig. 4.9 shows the nine H_i's used by Hueckel.

Note that in this procedure, the quantity N^2 directly gives us a measure of a quality of an edge. So, it is possible to use edge-acceptance criteria in which we require the step height h to be larger the greater the value of N^2.

4.1.4 Edge Detection Procedure Used in This Research

We have used Sobel operator shown in Fig. 4.3b for the research reported here. As was mentioned before, applying this operator leads to two output images that represent, respectively, the magnitudes and the orientations of the rate of change in brightness at each point in the image. The results for the image of Fig. 3.11a are shown in Figs. 4.10a and b. Note that the orientation image is extremely noisy. Also, note however that it is the magnitude value that determines whether or not an edge will be declared to be present at a pixel. So, the orientation values at a majority of the pixels shown in Fig. 4.10b will be of no consequence.

In general, higher level edge-based processing, such as might be needed for binocular fusion, becomes easier if the edges are only one pixel wide. In particular, the computational effort required by the Hough transform-based line detection algorithm of the next section is reduced if the edges are one-pixel wide. Therefore it is necessary that the process of thinning be applied to the edge magnitude images like the one shown in Fig. 4.10a. In our processing, we first apply Eberlein's algorithm [6] to thin the non-binary edge-magnitude images like Fig. 4.10a -- this algorithm, which is particularly well suited to the thinning of patterns that have continuously varying gray levels, accomplishes thinning by 'gathering' up the gray values towards their ridges. Thinned continuous gray-level edge images obtained in this manner are then thresholded to yield binary edge images, whose edges in general will not be one pixel wide. Subsequently, a binary thinning algorithm [1] is applied to yield one-pixel wide edges. An example of the output is shown in Fig. 4.11a; this edge output corresponds to the edge magnitude image of Fig. 4.10a.

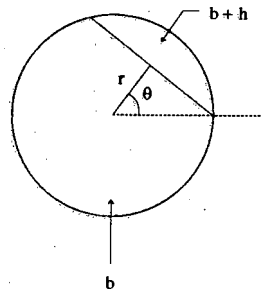


Figure 4.8: An ideal edge in a circular neighborhood.

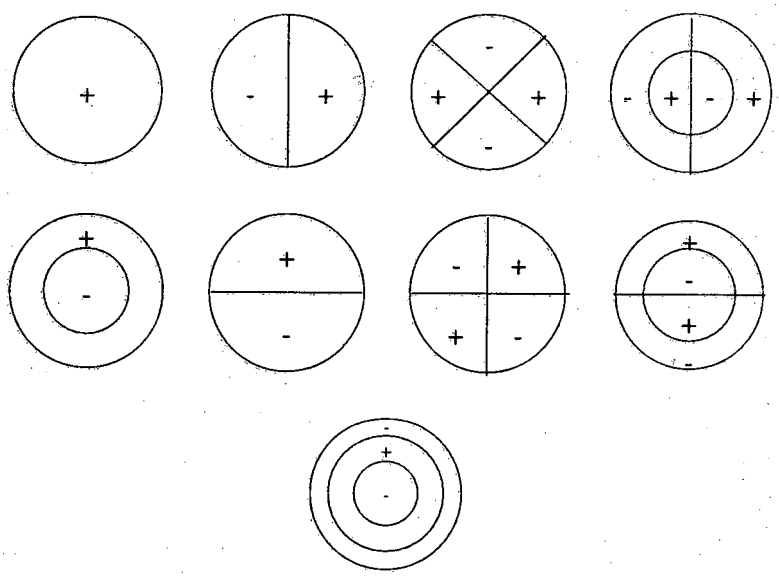
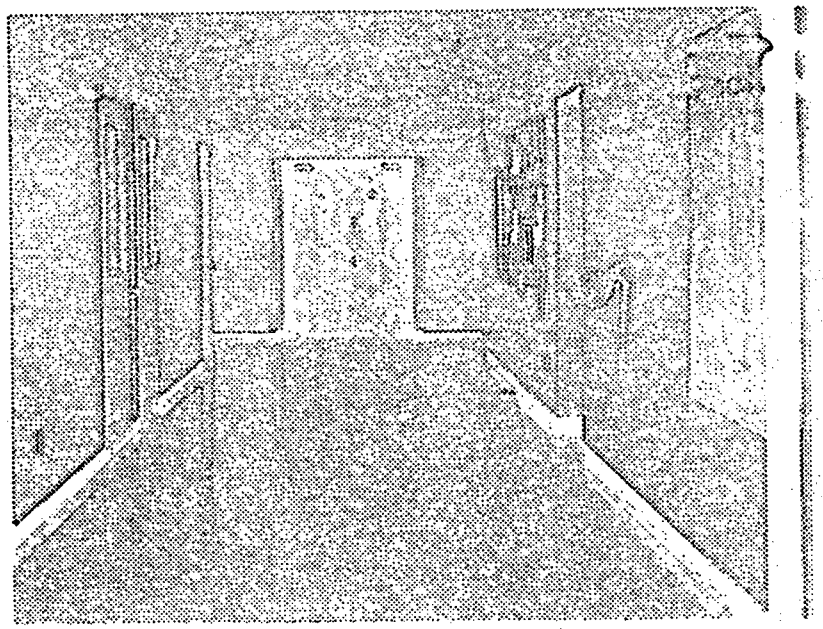
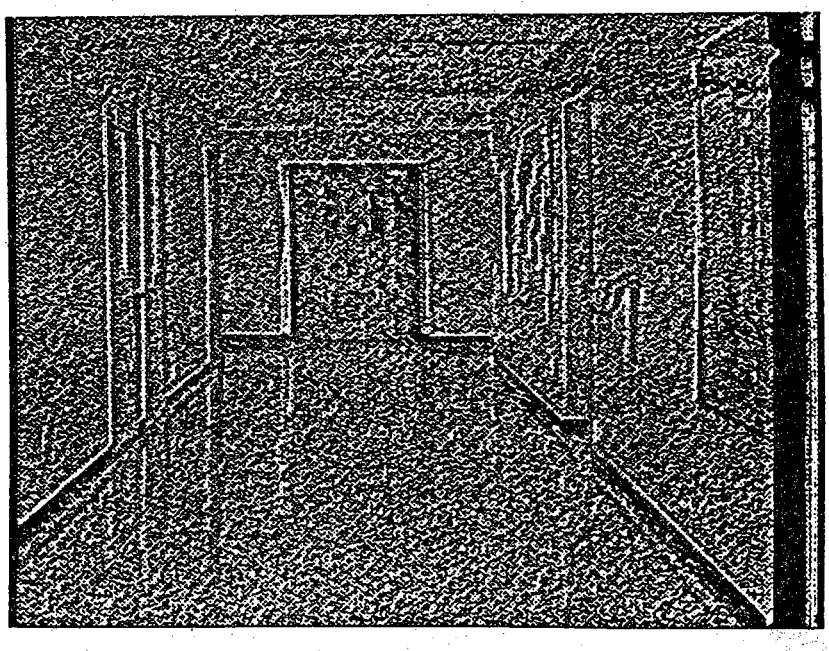


Figure 4.9: Basis functions for approximation in the Hueckel operator.

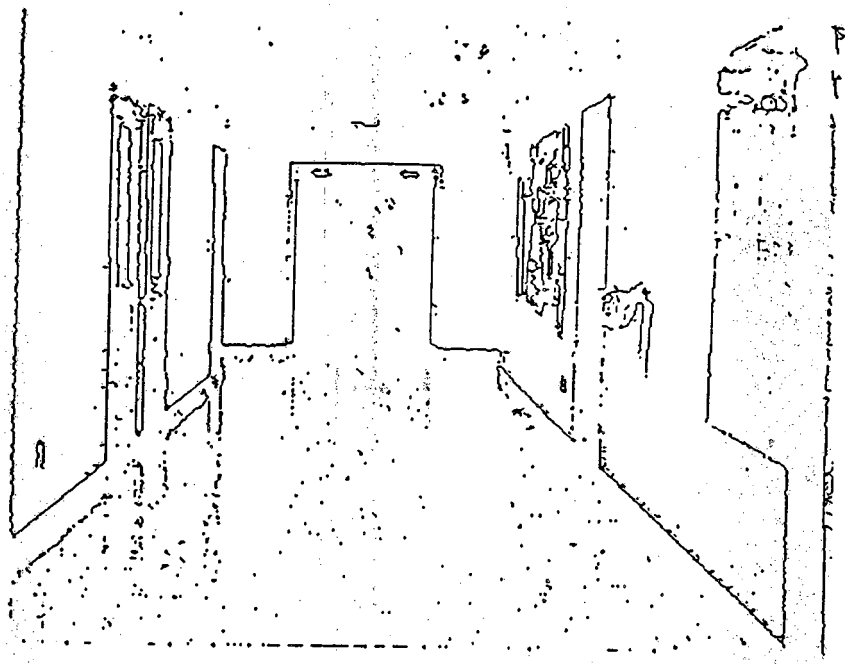


(a) Magnitude

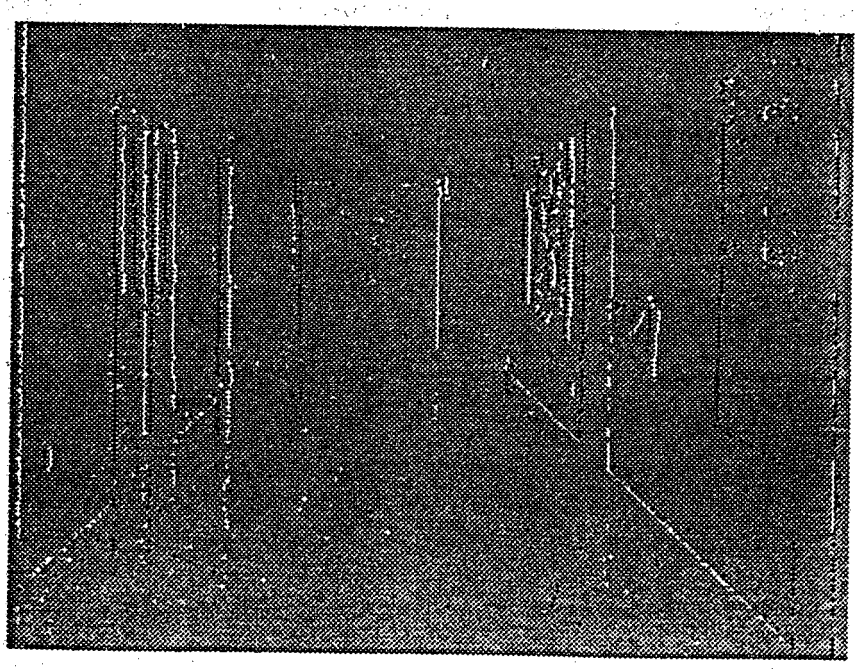


(b) Orientation

Figure 4.10: Output of the Sobel operator for the image of Fig. 3.10a.



(a) Magnitude



(b) Orientation

Figure 4.11: Output of the edge detecting process for the image of Fig. 3.10a.

The edge orientation image of Fig. 4.10b plays a useful role in the characterization of the thinned edges of Fig. 4.11a. For example, in Fig. 4.11b we have shown a result of one such characterization, obtained by using a three level quantization on the orientations. In Fig. 4.11b, we can distinguish between the edges that correspond to bright-to-dark transitions, shown as black edges, and the edges that correspond to dark-to-bright transitions, shown as white edges. Such edge characterization is important for stereo matching.

In Fig. 4.12, we have summarized the edge detection process used in this research.

4.2 LINE DETECTION

In low-level processing, ultimately our goal is to produce a symbolic map of a scene, the symbols corresponding to significant features that have been isolated and characterized by the values of their various attributes. The edge maps discussed in the previous section are still a numerical representation of a scene, in the sense that to use an edge map we must examine the value of the map at each pixel: if the value is high, the edge is present at that pixel, otherwise it is absent. Our next task is to group nearly-parallel and nearly-collinear edges into single entities called lines and then to represent each line as an individual entity, in other words as a symbol with its associated attributes and their values.

The line detection method that we implemented starts out with the edge image of Fig. 4.11a and groups the edges into lines using a Hough transform-based approach. Note that the Hough transform is a versatile tool for the detection and parameter estimation of general shapes. The main advantages of the Hough transform are its low sensitivity to noise and its ability to group shapes even when there are gaps present in the contour representing the shape. Unfortunately, there is a price to be paid for all this -- the computational effort required, especially if the intent is to also use the tool for estimating to high precision the various parameters associated with the detected shape. To reduce the computational effort and at the same time to maintain the desired accuracies in the estimated parameters of the lines, we use a two stage approach. Since we are only interested in the detection of vertical lines, corresponding to the vertical features of the hallways, we first use a fast projection technique to obtain approximate values for the locations of the vertical lines; subsequently the Hough transform is used in the vicinity of the lines so detected for a more precise fix on their locations. Approaches to line detection based on methods other than the Hough transform are reviewed [1] and [2].

In the rest of this section, we will briefly review the Hough transform approach to grouping pixels into linear features. For the purpose of this review we will assume that we wish to to apply the Hough transform to images like the one in Fig. 4.11a for the

---

1: Obtain the magnitude and orientation of the local edges from the output of a Sobel operator applied to the image I and store the information in images M and O respectively.

2: Thin the edges in the magnitude image M to make them at most one pixel wide. Select as local edges only the pixels above a given threshold storing them in image M'.

3: Quantize the values of the pixels in the orientation image O to three values corresponding to the edge orientations up, down and horizontal, obtaining image O'.

Figure 4.12: The local edge detection process summarized.

purpose of grouping approximately collinear edge pixels into the longest possible lines even when there are gaps between the edges.

Using the slope-intercept parameterization, a straight line in the image plane may be represented by the following form

$$y = mx + c \tag{4.17}$$

where m is the slope of the line and c the intercept of the line with the y-axis. Consider a point $\mathbf{P}_0$ (Fig. 4.13a) of coordinates (x_0, y_0) in the image space. We will assume that the point is located on a line L described by $L \equiv y = m_0 x + c_0$. Now an arbitrary line, of parameters m and c, which may not necessarily be the same line as shown in Fig. 4.13a, passing through the point $\mathbf{P}_0$ must satisfy

$$y_0 = m x_0 + c. \tag{4.18}$$

In the absence of any *a priori* knowledge about what line (or lines) the pixel $\mathbf{P}_0$ might belong to, we may take the position that this pixel should contribute a 'vote' to all the lines, each described by a pair (m,c), that satisfy Eq. (4.18). Computationally, this notion can be expressed by constructing an m,c space and depositing 'a unit vote' at all those points in the m,c space which satisfy Eq. (4.18). It is interesting to note that these points in the m,c space constitute also a straight line described by $c = -x_0 m + y_0$. Fig. 4.13b shows the parameter-space line, labeled L', corresponding to the single point $\mathbf{P}_0$ of Fig. 4.13a. We may thus say that the line L' is the Hough transform of the point $\mathbf{P}_0$. At the risk of sounding repetitious, we may also say that the point $\mathbf{P}_0$ votes for all the straight lines in the image space whose parameters fall on the line L' in the parameter space.

If we thus transform every non-zero pixel in Fig. 4.13a into the parameter space, we will obtain a collection of lines, all intersecting at a single point whose coordinates are (m_0,c_0). Stated equivalently, the maximum number of votes would be cast for the parameter pair m_0,c_0. A most important point to note here is that this outcome would not change significantly even the line L in Fig. 4.13a were broken. We therefore have a procedure for grouping pixels into lines even when the pixels themselves do not form a continuous line.

In practice, both the image space and the parameter space will be discretized. Therefore, the line L of Fig. 4.13a will consist of a finite number pixels whose coordinates will be close to but may not be exactly on the analytic line shown in the figure. This quantization effect will exhibit itself in the parameter space lines of Fig. 4.13c not intersecting at one point exactly. What's worse, there may not be a single unique maximum corresponding to the true maximum in the parameter space. Before we describe how to cope with this difficulty, we need to discuss the quantization of the parameter space.

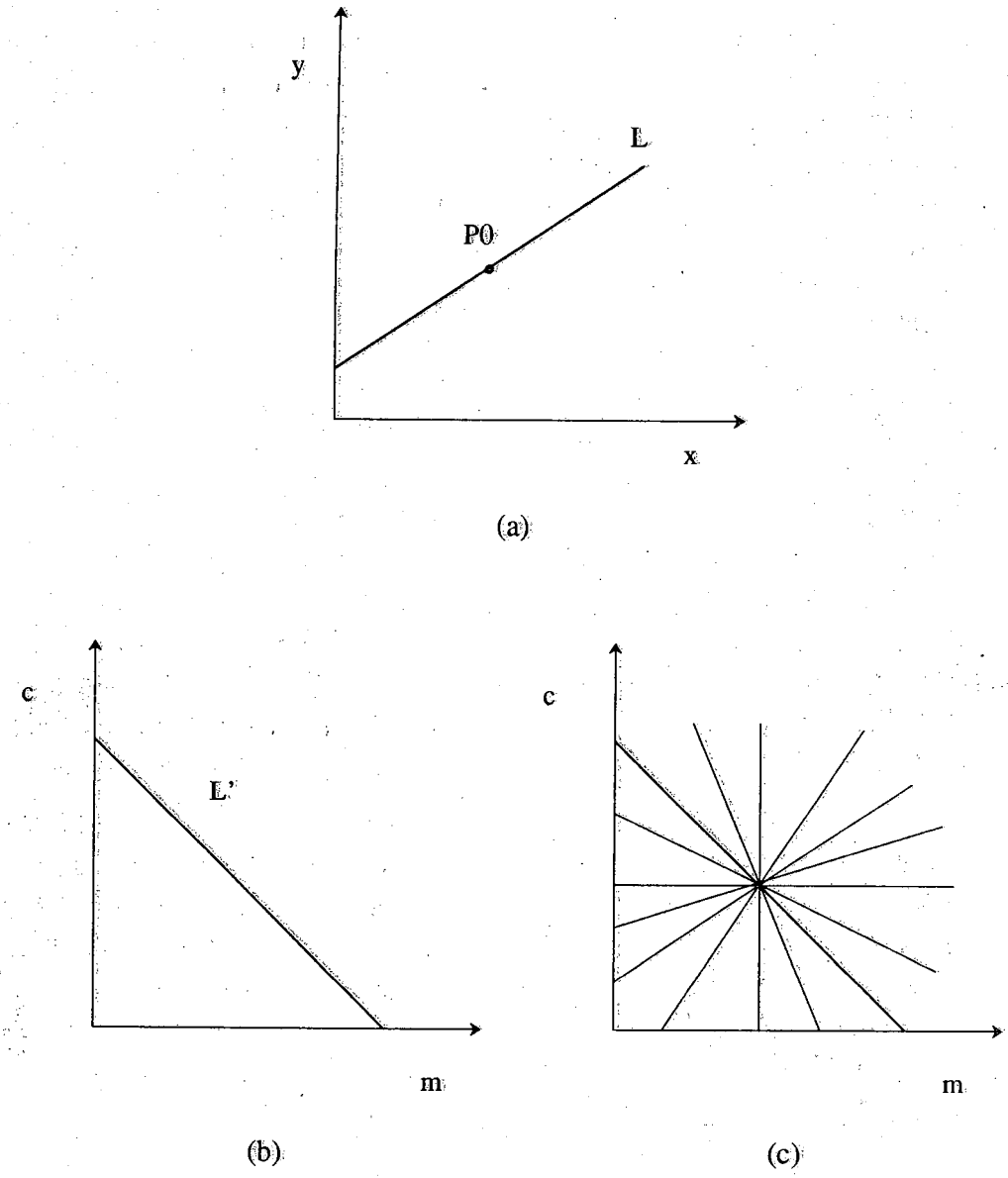


Figure 4.13: Mapping of a line into Hough space: (a) line in image space, (b) mapping of a single point in Hough space, (c) accumulation of mappings of collinear points in Hough space.

A nice intuitive interpretation can be given to the discrete version of the parameter space: We may view the quantized cells of the m,c space as buckets or accumulators. Then, when point $\mathbf{P}_0$ of Fig. 4.13a is under consideration, we compute the addresses of all the buckets which should receive a vote from $\mathbf{P}_0$ and then increment the count for each of these buckets. After all the non-zero pixels in the image space have been processed in this manner, we find local maxima of the bucket counts; the buckets corresponding to the local maxima yield the lines in the image space. This procedure is summarized in Fig. 4.14.

To deal with the deleterious effects of quantization of both the image and the parameter spaces, after the local maxima have been found, the parameters of the associated lines in the image space are computed by examining all the accumulated counts in the vicinity of each local maximum. The following formulas can be used to compute the m,c parameters of a line in the image space that gives rise to a local maximum in the parameter space:

$$m_i = \frac{\sum_{j=1}^{9} w_j m_j}{\sum_{j=1}^{9} m_j} \qquad c_i = \frac{\sum_{j=1}^{9} w_j c_j}{\sum_{j=1}^{9} c_j} \tag{4.19}$$

where w_j stands for the count in cell j of the 3×3 neighborhood of where a local maximum has been located; m_j and c_j represent the value of the parameters assigned to cell j. This approach finds the center of mass of the neighborhood using the vote count in each cell as the mass of that cell. When this procedure is used for estimating the parameters of a straight line, step 4 of the algorithm of Fig. 4.14 can be expanded as shown in Fig. 4.15.

The slope-intercept parameterization described above, first introduced in [10], presents two problems: an unbounded accumulator size and nonuniform quantization errors for the slope parameter. The accumulator size in unlimited, at least theoretically, because there are no upper bounds on the slope m and the intercept c. Since the value of the slope becomes large at an accelerated pace as a line in the image space approaches the vertical, in computer implementations it becomes necessary to discretize the m-axis nonuniformly; with uniform quantization a disproportionaly large number of the m-buckets would correspond to image space lines that are nearly vertical -- an unacceptable situation. To get around these two difficulties, Duda and Hart [11] have used for Hough transformation the parameterization of a straight line in terms of its shortest distance from the origin and the angle θ shown in Fig. 4.16. With this parameterization, the equation of a straight line in the image space is best expressed using polar coordinates. If we assume that the polar coordinates of an arbitrary point in the image space are (ρ,ϕ), then all the points lying on the straight line of Fig. 4.16a are described by the equation $\rho\cos(\phi - \theta) = r$. From this description, it should be clear that the Hough transform of a

---

1: Quantize the parameter space between appropriate maximum and minimum values for c and m.

2: Form an accumulator array $A(c, m)$ whose elements are initially zero.

3: For each point (x,y) in the feature image increment all points in the accumulator array along the line $c = -m\,x + y$.

4: The values of the accumulator array now provide a measure of the number of collinear points along a line of given slope and intercept. The highest local maxima correspond to the longest lines in the image.

Figure 4.14: Procedure to compute the Hough transform in the slope-intercept parametrization.

1: For each cell $A(c, m)$ in the the accumulator array compute the sum of its eight neighbors plus itself $S(c, m)$.

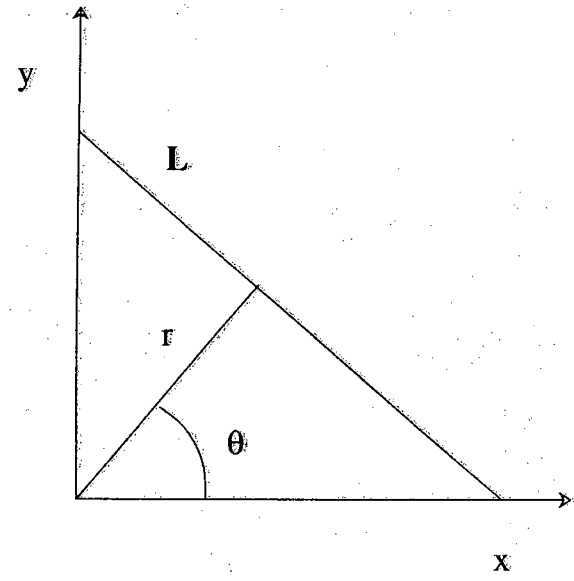
2: Find the local maxima in the array $S(c, m)$ that are above a given threshold.

3: Compute the parameters of the lines corresponding to the maxima of $S(c, m)$ as

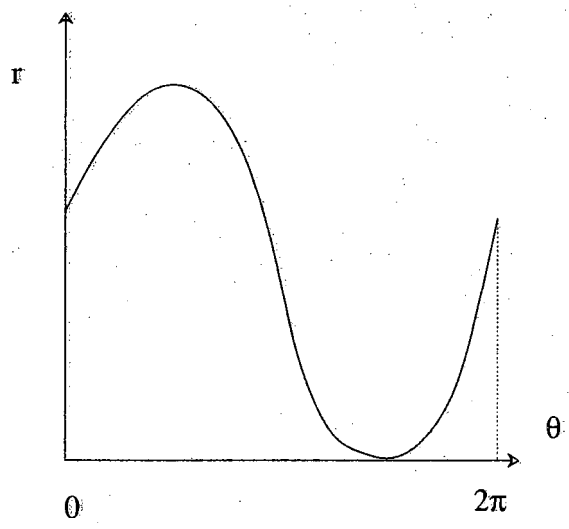
$$m_i = \frac{\sum_{m,c \in \mathbf{R}} A(c, m)\, m}{\sum_{m,c \in \mathbf{R}} A(c, m)} \qquad c_i = \frac{\sum_{m,c \in \mathbf{R}} A(c, m)\, c}{\sum_{m,c \in \mathbf{R}} A(c, m)}$$

where $\mathbf{R}$ is the 9-neighborhood in $A(c, m)$ of maxima i.

Figure 4.15: Procedure to compute the parameters of the lines in the image from the Hough transform accumulator array in the slope-intercept parametrization.



(a)



(b)

Figure 4.16: (a) The (r, θ) representation of a line. (b) Hough transform of a point under the (r, θ) parametrization: $x_i \cos\theta + y_i \sin\theta = r$.

single point in the image space is given by the sinusoidal curve shown in Fig. 4.16b. In other words, a single point in the image space votes for all the straight lines whose r, θ parameters lie on a sinusoidal curve like the one shown in Fig. 4.16b. Yet another approach was presented by Jain and Krig [12]. In their technique, a straight line in the image space is parameterized by its angle with the lower boundary of the image and the intercept made with any of the three sides shown in Fig. 4.17. Note that for measuring the intercept, a running index is established that starts at the upper left hand corner, goes clockwise around the image, to the lower left hand corner.

Due to the computationally intensive nature of straightforward implementations of Hough transformation, much effort in the past has focussed on discovering efficient implementations. Li, Lavin and LeMaster [13] have presented an algorithm which uses a hierarchical accumulator structure in the parameter space. Illingworth and Kittler [14] have introduced an approach that they call the Adaptive Hough Transform; this is an iterative algorithm in which a small sized accumulator is used and its parameter range redefined at each iteration to progressively focus on the correct parameters with increasing accuracy.

4.3 A VERTICAL LINE EXTRACTION PROCEDURE

We will now describe our procedure for grouping the edges in images like the one in Fig. 4.11a into vertical lines.

Note that a full-blown implementation of the Hough transform, especially when the images are of size 512×480, is too time consuming. For that reason we have adopted a two step procedure. In the first step, the subject of Section 4.3.1, approximate locations of the vertical lines are determined by a fast algorithm that carries out a column-wise summation of the pixels of the edge image. Subsequently, by using the procedure of Section 4.3.2, the Hough transform is applied separately to thin strips of the edge image, each strip consisting of a few columns of the image matrix around the vertical lines detected during the first step, for a more accurate calculation of the locations and the orientations of the vertical lines. In this fashion, the Hough transform calculations are speeded up not only because the number of pixels in an image strip is relatively small, but also because the geometry of an image-strip implies a reduced range of values in the parameter space.

4.3.1 Initial Line Location

The orientations of the cameras mounted on PETER is such that the scene vertical lines are approximately vertical in the images also. This allows us to make initial

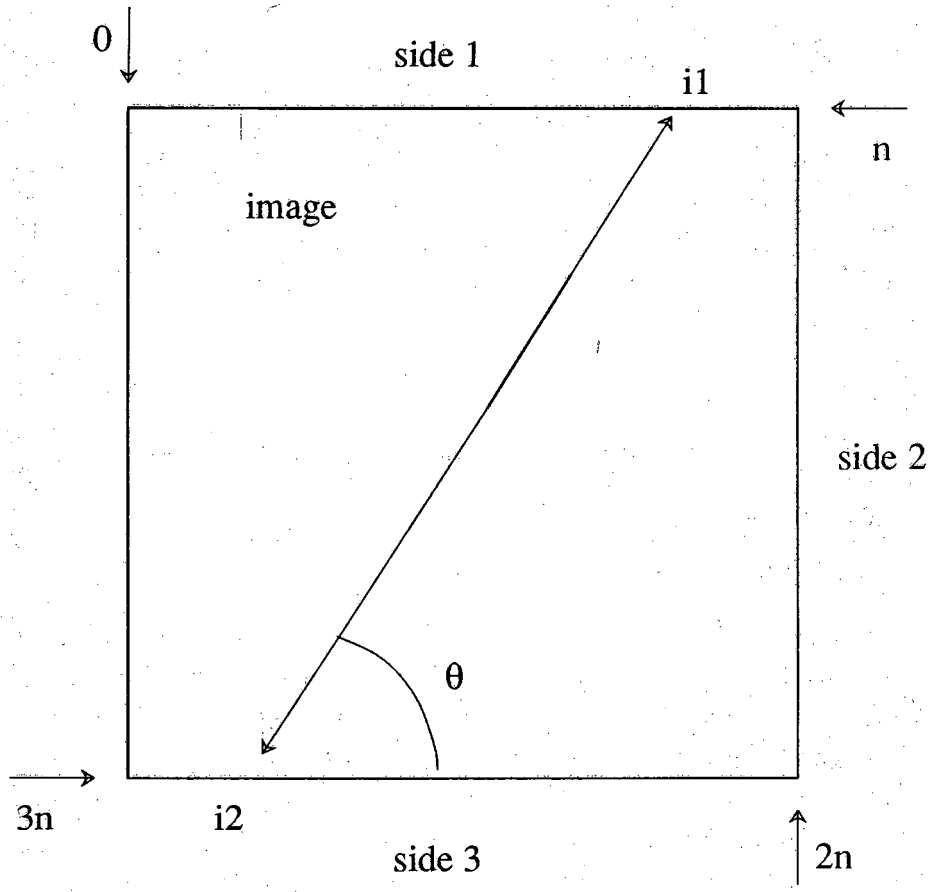


Figure 4.17: The (η, θ) representation of a line. The sides are numbered as in the figure, 1, 2 and 3 and the pixels on these sides are labeled with numbers from 0 to 3n. The orientation θ is measured from side 3 and η is the image border intersection with smallest label.

detection of the vertical lines by simply summing the image pixels in a column-wise fashion and looking for peaks in the projections thus obtained. In other words, we form a one-dimensional function $I(x)$ by summing an edge image $f(x, y)$ with respect to the index y: of the local edges image $f(x, y)$ as

$$I(x) = \sum_{y=1}^{480} f(x, y) \tag{4.20}$$

Performing this operation on the local edge magnitude and orientation images separately results in two one-dimensional functions, $I_m(x)$ and $I_o(x)$, respectively, for the projections of the magnitudes and of the orientations. Fig. 4.18a,b show these functions for the edge images of Fig. 4.11a,b.

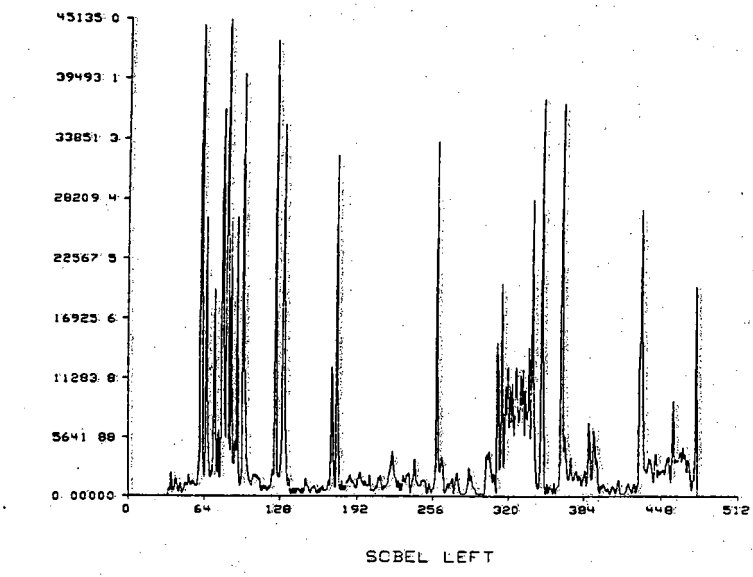
In the plot of Fig. 4.18a, it can be observed that a number of very strong peaks stand out against a noisy background. They correspond to the main vertical lines of the local edge image of Fig. 4.11a. For each of the strongest peaks in the projection-of-magnitudes plot of Fig. 4.18a, there is a corresponding positive or negative peak in the projection-of-orientations plot of Fig. 4.18b. The polarity of a peak in the orientation projection depends on whether the corresponding line in the edge image represents a bright-to-dark or dark-to-bright transition.

In order to detect the longest lines in the image from the magnitude projection, $I_m(x)$, the first step taken is to filter $I_m(x)$ in order to accentuate the largest of the peaks and to eliminate the uninteresting "low-frequency" variations. The filtering operation yields $F_m(x)$ via

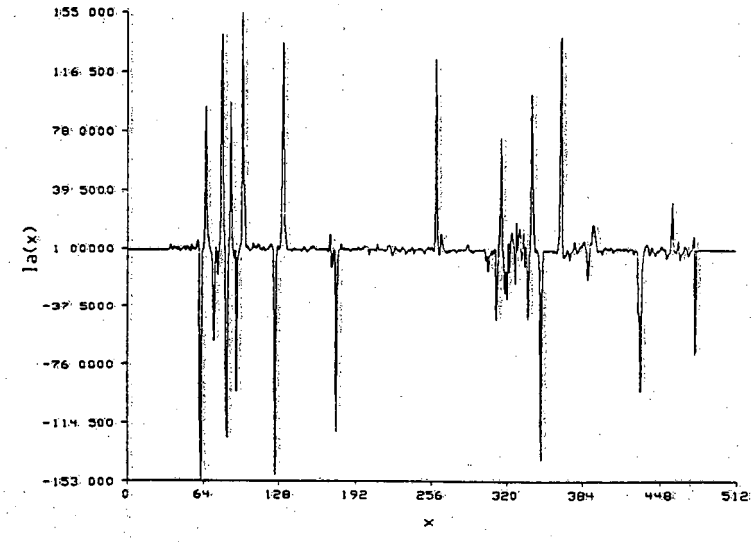
$$F_m(x) = I_m(x) - \frac{\sum_{i=-w}^{w} I_m(x+i) - \sum_{j=1}^{n} I_m(x_j)}{2w - n + 1} \tag{4.21}$$

for *all* $j = 1,...,n$ and for *all* x; $x_j, x \in [x-w\ ,\ x+w\ ]$ and $I_m(x_j) \geq I_m(x)$

That is, $F_m(x)$ is the difference between the function $I_m(x)$ and its local average computed over a window of size $2w + 1$ once the n highest values of $I_m(x)$ are removed. Note that within each window $(x - w\ ,\ x + w)$, the variable n is equal to the number of values of I_m which equal or exceed the value $I_m(x)$. Since the second term on the right hand side is the average of all the I_m values within the window that are strictly less than $I_m(x)$, we make sure that the process of suppressing low-frequency variations does not lead to the peaks distorting one another. A plot of $F_m(x)$ for the function $I_m(x)$ of Fig. 4.18a is shown in Fig. 4.19a. After computing $F_m(x)$, its local maxima are determined and their average height computed. The peaks selected as corresponding to the longest vertical lines in the edge image are the ones whose heights exceed a certain threshold times the average local maxima height. Using this procedure, the peaks detected are



(a) $I_m(x)$



(b) $I_o(x)$

Figure 4.18: Projection functions for the images of Fig. 4.11.

shown in Fig. 4.19b for the $F_m(x)$ of Fig. 4.19a.

If the lines were perfectly vertical in the edge image, the peaks detected in $F_m(x)$ would have one-pixel widths and the location of a line would be exactly the x coordinate of its corresponding peak. But if the cameras are not exactly horizontal, the direction of the columns in the image plane will not coincide precisely with the world vertical direction. As a result, in general, a vertical line in the scene will give rise to an image line that spans several columns. Consequently, the peaks in $F_m(x)$ will, in general, be "thick" peaks whose middle points may be taken as rough estimates of the locations of the vertical lines -- these locations to be refined later by the Hough transform technique, where the actual slopes of the lines would also be computed.

Each peak selected in $F_m(x)$ is assigned a *polarity* which is equal to the polarity of $I_o(x)$ at the middle point of the peak. This polarity value is subsequently used the stereo matching algorithm.

The procedure described in this subsection for detecting lines is summarized in Fig. 4.20.

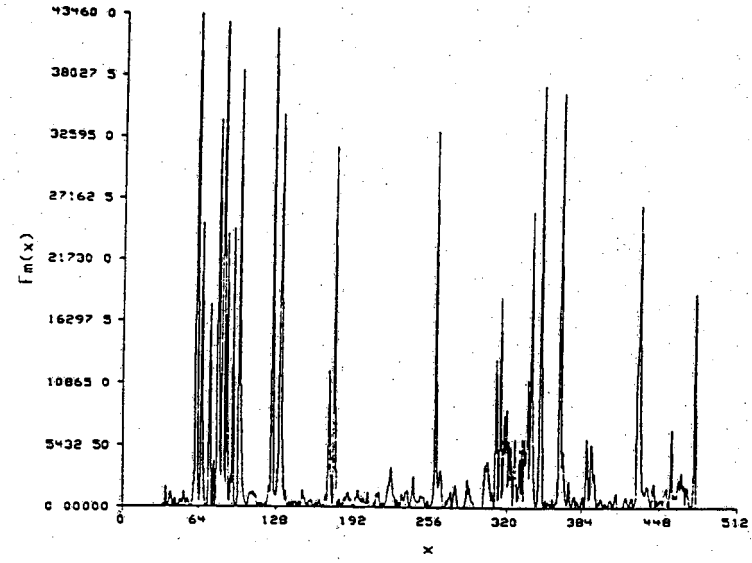
4.3.2 Improving the Estimates of Line Parameters

The final parameter estimation is obtained using a Hough transform technique that considers one line at a time as detected by using the technique of the preceding subsection. To avoid problems with the unbounded slopes of lines that are perfectly vertical, lines are represented by equations of the form $x = m\,y + c$, instead of the more familiar $y = m'x + c'$. Therefore, in the discussion to follow, the parameter m will represent the tangent of the angle that a line forms with the y-axes and c the x-intercept.

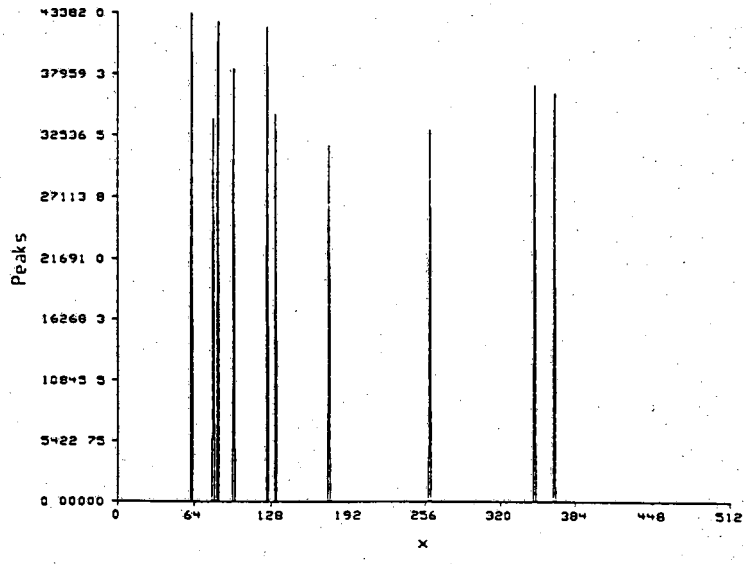
In accordance with our earlier discussion, each line detected from the peaks of $F_m(x)$ is surrounded by a rectangular strip and a Hough transform computed of all the pixels in this strip. Since we are only interested in long lines and since for long lines the slopes and intercepts can only take a small range of values, the parameter space is constructed to reflect this fact and divided into cells. Delimiting the parameter values in this manner contributes significantly to the speedup of Hough transformation.

After the Hough transform is performed over a given strip of the edge image, the cells of the parameter space are searched for the most significant peak and the corresponding parameters computed by using the procedure outlined in Fig. 4.15.

A problem arises when there is an overlap between the rectangular strips corresponding to two or more lines detected by the projection procedure of Section 4.3.1. In this case, prior to Hough transformation, we take the union of all such overlapping



(a) $F_m(x)$



(b) Peaks detected

Figure 4.19: Final projection function (a) and peaks selected (b).

1: Compute the local edges magnitude and orientation images M' and O' following the steps of the algorithm in Fig. 4.12.

2: Add the values in each column of M' and O' obtaining the functions I_m and I_o respectively.

3: Filter I_m obtaining F_m as

$$F_m(x) = I_m(x) - \frac{\sum_{i=-w}^{w} I_m(x+i) - \sum_{j=1}^{n} I_m(x_j)}{2w - n + 1}$$

for all j = 1,...,n and for all x; $x_j, x \in [x-w\ ,\ x+w\ ]$ and $I_m(x_j) \geq I_m(x)$

4: Compute all the local maxima in F_m and find the average value of F_m at those maxima a.

5: Select the maxima at which the value of F_m is higher than a given threshold times the average a. Assign to these selected maxima a *polarity* depending on whether the value of I_o at these points is over or under the background value.

Figure 4.20: Algorithm for computing the estimated location of vertical lines.

rectangular strips, and modify the parameter space to reflect the larger range of values that can now be taken by the slope and the intercept parameters corresponding to long lines in this larger piece of the image. Of course, now the parameter space is searched for not just one largest peak, but a number of largest peaks equal in number to the number of strips merged.

This approach to line parameter refinement is summarized in Fig. 4.21. Fig. 4.22 presents the overall approach to the detection and extraction of vertical lines in a gray level image. In the table in Fig. 4.23, we have shown both the initial locations of the vertical lines and their final locations and slopes for the image of Fig. 3.11a. A composite formed by superimposing on the original image the ideal lines whose parameters were calculated is shown in the image in Fig. 4.23.

1: Form a list L with lines found using the procedure of Fig. 4.20 ordered by their horizontal intercept estimate.

2: For each line L_i in list L select a vertical strip S_i in the image centered at the estimated horizontal intercept of L_i in which the line must be contained.

3: Until the list is empty perform steps 4-11.

4: Remove lines $L_i, \cdots, L_j$ from the head of list L until the strip S_j of the last line removed L_i does not overlap with the strip S_{j+1} of the new head of the list, or until the list is empty.

5: Select the area of the image S corresponding to the intersection of the strips $S_i, \cdots, S_j$ of the lines removed in step 4 from L.

6: Compute the parameter range compatible with the new area S and the original areas of the individual lines $S_i, \cdots, S_j$ as follows: the horizontal intercept range has to cover the horizontal extent of S and the slope range has to cover the union of the ranges of each individual line.

7: Construct an accumulator A that covers the range of parameters determined in step 6 to a given accuracy (the range covered by each cell in the accumulator).

8: Perform the Hough transform of the image area S into the range of parameters covered by the accumulator A.

9: Perform the steps 10 and 11 once for each line in S ($j - i$ times).

10: Select the most prominent peak in the accumulator and compute the parameters associated with it using the method of Fig. 4.15. This are the final parameters of the corresponding line.

11: Set to zero all the cells associated with the peak found in step 10.

Figure 4.21: Procedure to obtain the parameters of the vertical lines in the image using a Hough transform technique.

1: Compute the magnitude and orientation local edge images M' and O' using the Sobel operator thinning and thresholding as described in the algorithm of Fig. 4.12.

2: Compute the approximated location of vertical lines and their polarity from the projections I_m and I_o of M' and O' respectively as described in the algorithm of Fig. 4.20.

3: For each line L found in step 2 perform steps 4 and 5.

4: Compute the local Hough transform around the estimated location of line L as described in the algorithm of Fig. 4.21.

5: Search the accumulator obtained in step 5 for the most prominent peak and compute the parameters of its corresponding line as described in the algorithm of Fig. 4.15. The polarity of the line is taken as the one found in step 2.

Figure 4.22: Description of the feature extraction procedure.

Table 4.1: Results of the line extraction.

Projection	Hough	
Intercept	Intercept	Slope
62	62.302862	-0.002427
80	80.303229	-0.002400
84	84.300000	-0.002225
97	98.099719	-0.002900
125	125.000909	-0.000307
132	131.918776	-0.000311
176	178.201304	-0.007432
260	260.076140	-0.000168
349	346.801208	0.007309
366	364.798201	0.002714

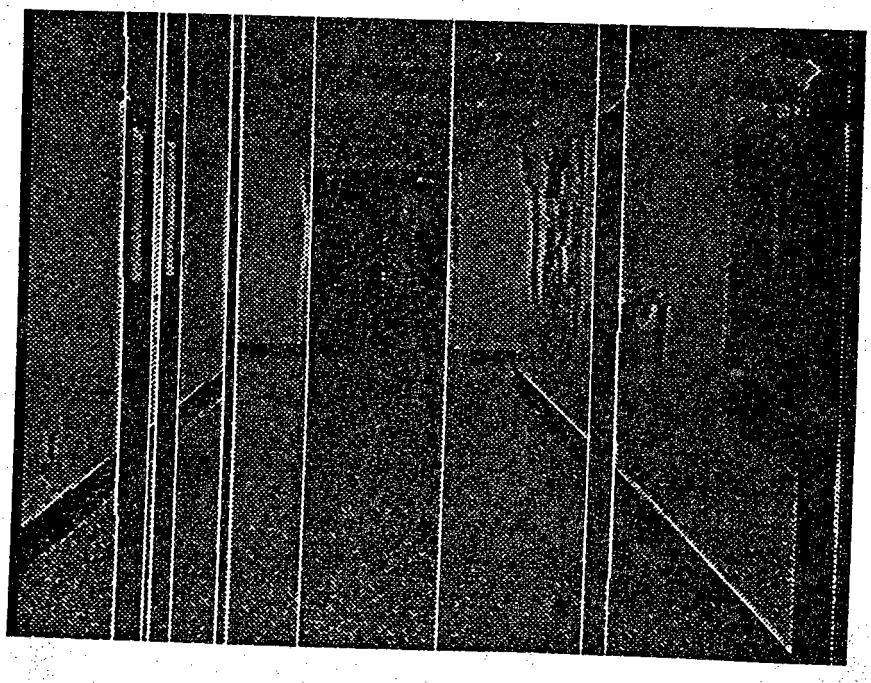


Figure 4.23: Final lines detected in the image of Fig. 3.10a.

REFERENCES

[1] A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, Academic Press, 1980.

[2] D. H. Ballard and C. M. Brown, *Computer Vision,* Prentice-Hall, 1982.

[3] R. Nevatia, *Machine perception,* Prentice-Hall, 1982.

[4] D. Marr and E. Hildreth, "Theory of Edge Detection", *Proceedings of the Royal Society of London,* B207, pp. 187-217, 1980.

[5] W. E. L. Grimson, *From Images to Surfaces,* M.I.T. Press, 1981.

[6] R. B. Eberlein, "An Iterative Gradient Edge Detection Algorithm", *Computer Graphics and Image Processing*, vol. 5, pp. 245-253, 1976.

[7] R. Nevatia and K. R. Babu, "Linear feature extraction and description", *Computer Graphics and Image Processing*, vol. 13, pp. 257-269, 1980.

[8] M. H. Hueckel, "An Operator that Locates Edges in Digital Pictures", *Journal of the ACM,* Vol. 18, pp. 113-125, 1971.

[9] M. H. Hueckel, "A Local Visual Edge Operator Which Recognizes Lines", *Journal of the ACM,* Vol. 20, pp. 634-647, 1973.

[10] P. V. C. Hough, "Method and means for recognizing complex patterns", U.S. Patent 3,069,654, 1962.

[11] R. O. Duda and P. E. Hart, "Use of the Hough transformation to detect lines and curves in pictures", *Communications of the ACM*, vol. 15, pp. 1-15, 1972.

[12] A. N. Jain and D. B. Krig, "A robust Hough transform technique", *Vision*, vol. 4, pp. 1-5, 1987.

[13] H. Li, M. A. Lavin, and R. J. LeMaster, "Fast Hough Transform", *Proc. 3rd Workshop Computer Vision: Representation and Control,* pp. 75-83, 1985.

[14] J. Illingworth and J. Kittler, "The adaptive Hough transform", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-9, pp. 690-698, 1987.

CHAPTER 5

STEREO MATCHING AND TRIANGULATION

As was mentioned at the beginning of Chapter 4, our mobile robot uses two methods for self-location: one based on a stereoptic fusion of the images from the two cameras on board the robot, and the other based on a comparison of a single image from one of the cameras with an expectation map derived from a CAD model of the hallways. In this chapter we will discuss the first of these two methods and relegate the second to the next chapter.

During the last several years a number of algorithms have been developed for depth perception via passive stereopsis. To be sure, even before the current surge of interest in the topic -- driven primarily by a desire to endow robots with 3D perception -- much research had been done on the subject in the context of photogrammetry. Interestingly, due to the nature of the problem domains, the stereopsis algorithms for robotic applications have turned out to be different from those for photogrammetric applications. Due to the nearness of distances involved (compared to the camera-to-scene distances in photogrammetry) and large variations in depth possible over these distances, robotic images tend to suffer more from occlusion and scale compression-expansion effects; this makes it virtually impossible to use in robotics the area-based methods of stereopsis developed for photogrammetry. Therefore, the algorithms developed for robotic applications have tended to depend more on the matching of features, such as the vertical lines discussed in the preceding chapter, for the calculation of depth.

In the rest of this chapter, we will first very briefly discuss the different approaches to stereopsis. Subsequently, we will present the algorithm we have used for pairing up the prominent vertical lines extracted from the two images. Finally, we will show how the equation of a 3D scene line can be obtained from the matched vertical lines extracted from the two images of a stereo pair.

5.1 A BRIEF REVIEW OF STEREOPSIS

The fundamental problem of stereopsis is the establishment of corresponding pixels from the two images of a stereo pair. Ideally, we would like for every pixel not subject to occlusion in the left image a corresponding pixel from the right image, corresponding in the sense that the two pixels belong to the same scene point. Despite its straightforward nature (and despite the apparent ease with which we humans exercise stereoptic vision), the correspondence problem has proved to be exceedingly difficult to solve. Of course, after the correspondences are established, the disparities can be calculated, the disparity at a pixel in the left image being equal to $d_1 - d_2$, where d_1 is the distance of the left-image pixel from the optic axis of the left camera and d_2 the distance of the corresponding right-image pixel from the optic axis of the right camera. By using triangulation formulas, disparities translate directly into depth information.

Broadly speaking, the algorithms that seek to solve the correspondence problem fall into two categories: the area-based algorithms, developed originally for photogrammetric applications, and the more-recent feature-based algorithms developed for robotic applications. Each of these categories will now be briefly reviewed.

5.1.1 Area-Based Stereo

The correspondence problem is solved by correlating a patch of the left image surrounding the pixel whose corresponding right-image pixel is sought with comparably sized patches from the right image. The location in the right image yielding the maximum value for the correlation supposedly corresponds to the sought right-image pixel.

One of the major shortcomings of area-based stereo algorithms is their inability to cope with scale expansion and contraction effects common to imagery in robotics. To explain, consider an object surface that is slanted with respect to the optic axes of the two cameras, meaning the surface is not perpendicular to the optic axes. In general, the scale associated with the perspective projection of this surface on the two cameras will be different, because the slant of the surface with respect to the two optic axes will be different. Such scale differences lead to noisy and erroneous correlations, making difficult the detection of correspondences.

Another problem with area-based methods is their excessive sensitivity to brightness variations. Since the peak of a correlation is proportional to the product of gray levels in the two image patches taken from the left and the right images, scene surfaces of the same texture quality will yield very different results depending on the extent of shadowing and illumination. Much more so than the feature-based methods, area-based methods also appear to be more susceptible to occlusions.

Some area-based algorithms have been employed in automatic cartography applications; however, all practical systems require the intervention of human operators to guide and correct them. Some area based systems are described in [2,3].

5.1.2 Feature-Based Stereo

In feature-based stereo, we first extract features such as edges, intersection of lines, zero-crossing contours after the images are filtered with Laplacian-of-Gaussian operators, points with large gray-level variations in all directions, etc. Correspondence problem in this case consists of pairing up the features from the left and the right images.

Advantages of feature-based methods include their lower sensitivity to brightness variations in images and faster computation, since the number of features extracted from an image is usually not very large. In addition, when subpixel techniques are used for calculating the locations, feature-based techniques also tend to yield more accurate depth values compared to area-based algorithms. Feature-based methods do suffer from one disadvantage: the resulting depth maps tend to be sparse since depth values can only be computed at those pixels where the features are located.

Given a feature from, say, the left image, finding its corresponding feature from the right image evidently involves some search from among the candidate right-image features selected on the basis of their similarity to the left-image feature. Various constraints can be invoked during this search (and also for the formation of the pool of candidate features), the two commonly used being the epipolar constraint, and the smoothness-of-disparities constraint. While the latter constraint is obvious for what it implies, we will now say a few words about the former constraint.

The epipolar constraint says that given a pixel on a feature in, say, the left image, its corresponding pixel in the right image must lie on what's called the *epipolar line*. To explain, recall that under the thin-lens approximation there exists a line-of-sight for each each pixel in an image, the line-of-sight passing through the pixel and the lens center. In Fig. 5.1, we have shown a pixel P_1 and its corresponding line-of-sight. A perspective projection of this line-of-sight onto the image plane of the right camera is the epipolar line in the right image for pixel P_1 of the left image. In other words, an epipolar line for a left-image pixel is the right-camera image of the line-of-sight corresponding to the left-image pixel. The epipolar constraint says that the right-image correspondent of the left-image pixel P_1 must lie on the epipolar line shown in the figure. By analyzing the geometry associated with epipolar lines, it can be shown that when the optic axes of the two cameras are parallel, the epipolar lines must also become parallel to the camera

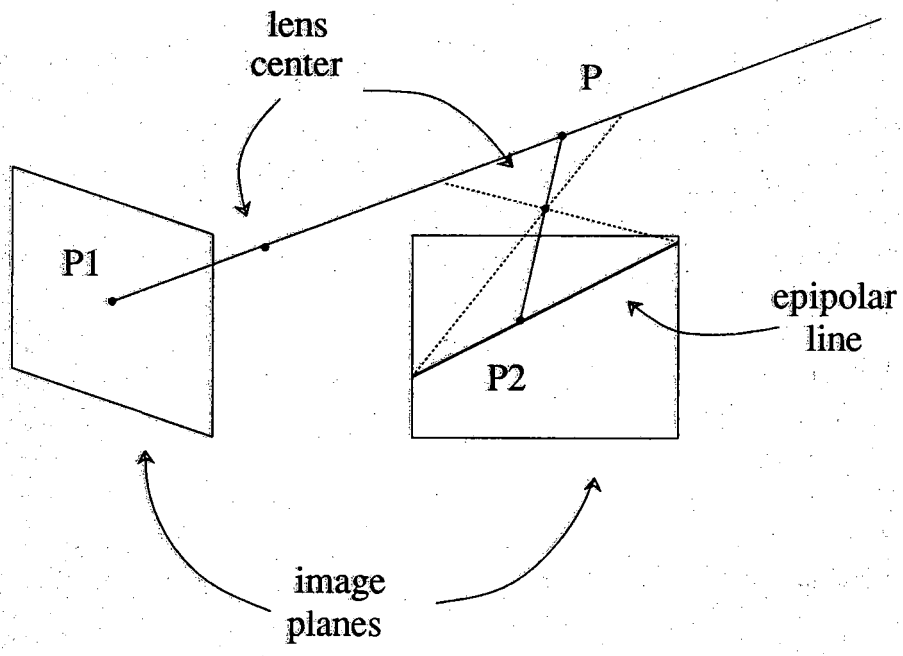


Figure 5.1: Epipolar geometry.

baseline, which is the line joining the two lens centers. When this baseline is parallel to the horizontal scan lines of the cameras, given a pixel on a particular scan line of the left image, its right-image corresponding pixel must be located on the same scan line of the right image. The plane formed by the line-of-sight for P_1 shown in Fig. 5.1 and the lens-center of the right camera is called the *epipolar plane*. Equivalently, the epipolar plane for a pixel from, say, the left image is defined as the plane containing the pixel and the two lens centers. The intersection of the epipolar plane and the right camera image plane defines the epipolar line for the left-image pixel in question.

Some feature-based systems are described in [1,4,5,6]. For a survey-type discussion, the reader is referred to [3, 7].

5.2 A MATCHING ALGORITHM

Long vertical lines -- they do not have to be exactly vertical -- as extracted by the method of Chapter 4 are used as features in a feature-based stereo we have implemented for our mobile robot. We have implemented a variation of the algorithm described in [4] for pairing up the lines in the two images.

The matching algorithm uses only two pieces of information about each line: the x-intercept and the polarity. The polarity indicates whether the line represents a dark-to-bright or a bright-to-dark transition in the original image. The polarity, under normal circumstances, is a viewpoint independent property and is therefore useful in disambiguating different possible right-image matches for a given vertical line from the left image. The x-intercept used in the matching process is not the value found by the Hough transform step but the value estimated initially from column-wise projections (See Chapter 4). There is an interesting reason for that: The x-intercept value as generated by Hough transformation corresponds to the intersection of an infinite line, corresponding to the grouping of edges found in the edge image, with the x-axis. On the other hand, the x-intercept value provided by column-wise projections represents approximately an average value of the x-coordinates associated with the pixels in the grouping. Given two different lines of different slopes, the former x-intercept can reverse the left-to-right order of the appearance of the lines, whereas that's less likely to happen with the latter type of x-intercept. To explain, assume we have two line segments, S_1 and S_2, as shown in Fig. 5.2, one perfectly vertical and the other with some slope to it. If we had to associate a left-to-right order with these two lines, our answer would be erroneous if we used the x-intercepts generated by Hough transformation, as the transformation would yield $X1$ and $X2'$ for the intercepts. On the other hand, a left-to-right order inferred from a column-wise projection, which may look like Fig. 5.3, will be based on $X1$ and $X2$ and would be correct.

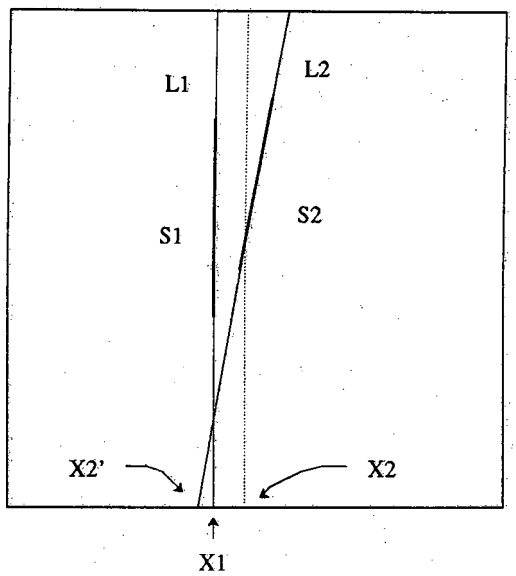


Figure 5.2: Apparent inversion in the order of the lines. The Hough transform will find $X\,1$ and $X\,2'$ as intercepts.

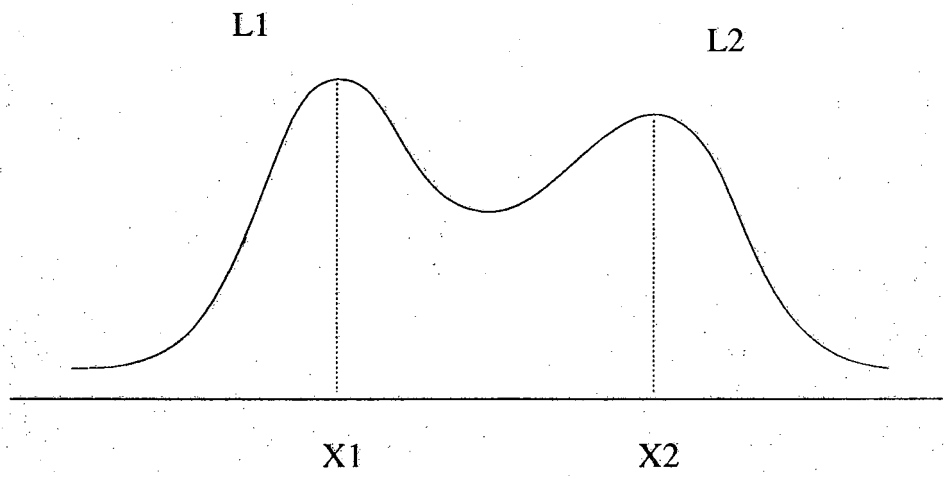


Figure 5.3: Lines of Fig. 5.2 as detected by the projection algorithm.

The matching algorithm consists of the following two steps:

1. For each line in each of the two images, construct a pool of candidate lines from the other image.
2. From the pool of candidate lines, construct a best possible match for each of the lines, best in the sense that the disparity associated with each line best agrees with the disparities associated with the neighboring lines.

The set of possible matches for a line A_i in the left image is the set of lines $\{B_j\}$ in the right image such that each B_j satisfies the following conditions:

a. The x-intercept, x_{A_i}, of the line A_i in the left image and the x-intercept, x_{B_j}, of the line B_j in the right image should satisfy $x_{B_j} \geq x_{A_i}$, assuming that all the x-intercepts are measured from the lower left hand corners of each of the image frames.

b. The disparity associated with a potential match is smaller than a threshold corresponding to the furthest distance at which scene points may exist. Mathematically, this translates into the condition: $\mid x_{A_i} - x_{B_i} \mid \leq d_{\max}$.

c. The polarity of both lines, A_i and B_j, is the same.

The idea behind the first condition is shown graphically in Fig. 5.4. The image of a scene point in the left camera will appear to the left of the image of the same point in the right camera if the optic axes of the cameras cross or intersect beyond the location of the scene point. In our experiments, the camera optic axes are nearly parallel, hence condition (a) above. Condition (b) is warranted by the fact that there will always be an upper bound on the distance to the objects in a scene. Condition (c) follows from the viewpoint invariance of line polarities.

In the second step of the algorithm, the lines from the left and the right images are paired up so that the resulting disparity field is the smoothest. There is psychophysical evidence that the human visual system does the same. Consider, for example, the case shown in Fig. 5.5, where the left and the right images contain three features each; each feature is represented symbolically by a square in the figure. If we pair up the leftmost feature in the left image with the leftmost feature in the right image, the center feature in the left image with the center feature in the right image, and the rightmost feature in the left image with the rightmost feature in the right image, we get a flat interpretation of the scene, as marked by the small circles. On the other hand, if the leftmost feature in the left image is matched with the rightmost image in the right image, the center feature in the left image with the leftmost feature in the right image, the rightmost feature in the left

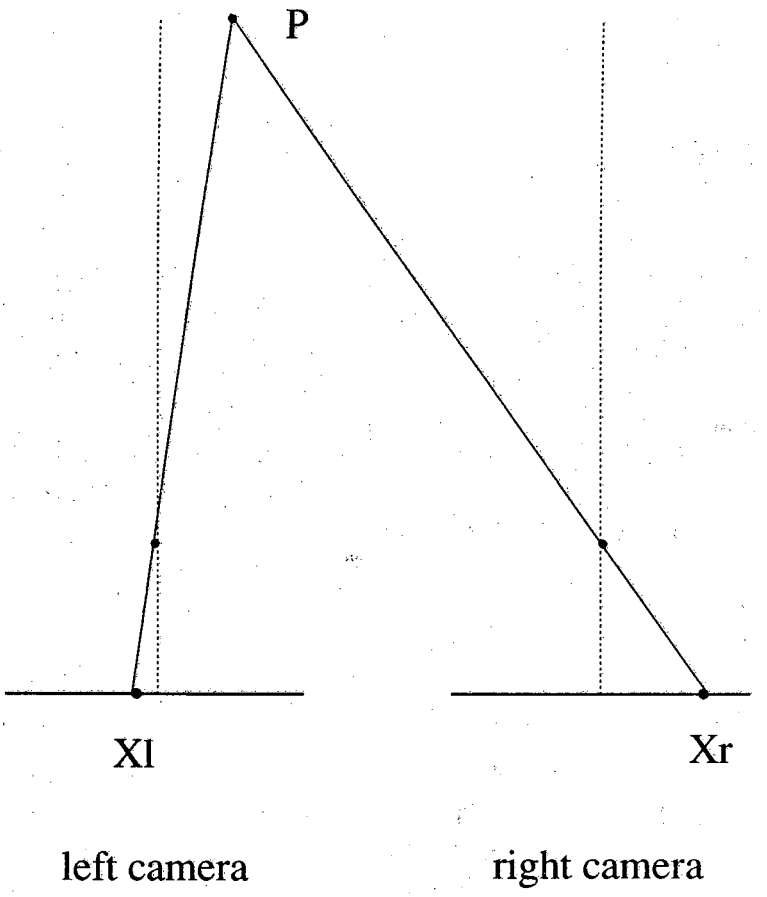


Figure 5.4: Relative location of the images of the same point.

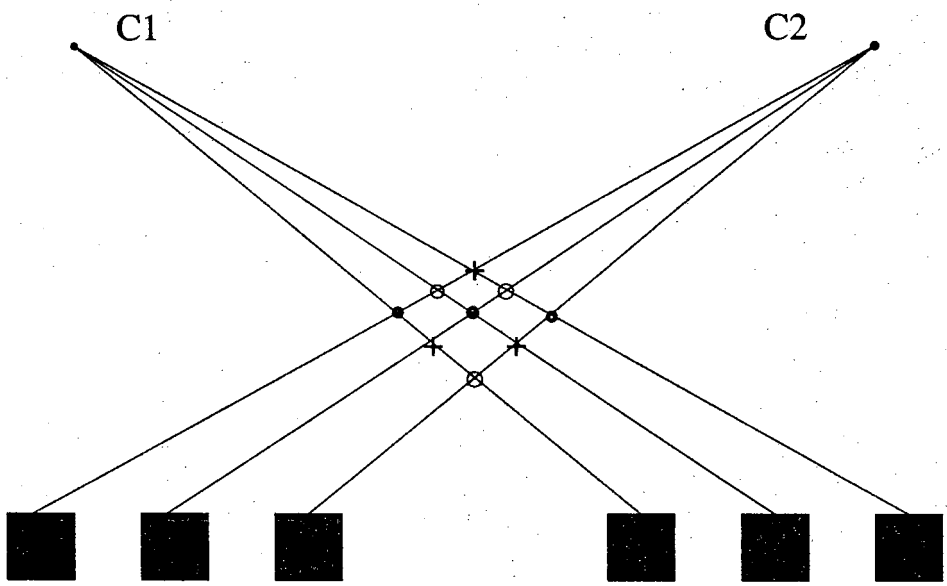


Figure 5.5: Example of ambiguity.

image with the center feature in the right image, we get an interpretation shown by the crosses in the figure. Other ways of pairing up the lines from the two images yield other interpretations in the scene. Everything else being equal -- such as all the features being visually identical -- out of all such interpretations, the human visual system selects the flattest interpretation, the one marked by the small circles in Fig. 5.5.

In accordance with the procedure advanced in [4], a smoothness constraint for computer implementation can be derived from the rationale that at each vertical line we want the disparity to take a minimum value provided such a minimum value is consistent, from the standpoint of smoothness, with the disparities at the neighboring vertical lines. In other words, we want to pair up the vertical lines in such a manner that for each match the resulting disparity is minimum under the condition that the difference in disparity for the match and the disparities at all the neighboring matches is a minimum. Let d_{AB} be the disparity associated with matching line A from the left image with line B from the right image. For the purpose of imposing the smoothness constraint, let's define the vicinity of line A as consisting of all the x-intercepts that fall within a $2d_{\max}$ interval $(x_A - d_{\max}, x_A + d_{\max})$. We will denote this interval by the symbol $W(A)$. The symbol $W(B)$ will denote a similar interval around the line B in the right image. Now let C be another vertical line in the $W(A)$ vicinity of A in the left image and let D be a potential match for C -- D can be any line from the pool of candidate line matches for C. Let d_{CD} be the disparity associated with matching C with D. Out of all the possible D's, let's select that D which yields a minimum value for d_{CD}. Clearly then the difference $d_{AB} - d_{CD}$ is a measure of the smoothness implied by the match (A,B) vis-a-vis the match (C,D), the latter being the smallest possible disparity match for C. Such a measure of smoothness needs to be integrated over all possible C's in the $W(A)$ neighborhood of the line A. The following expression shows this integrated measure together with a symmetrical component generated by applying similar arguments to line B in the right image:

$$v(A,B) = \sum_{C \text{ in } W(A)} \min_{D} \frac{|d_{AB} - d_{CD}|}{card(W(A))} + \sum_{C \text{ in } W(B)} \min_{D} \frac{|d_{AB} - d_{CD}|}{card(W(B))} \qquad (5.1)$$

where $card(W(A))$ represents the number of matches over which the sum is defined.

A computer program is easily written to select final matches for the vertical lines such that the measure in Eq. (5.1) is minimized for each match. For each final pairing (A,B) selected by the program, the following conditions must be satisfied:

for every other possible match D for A, $v(A,B) < v(A,D)$,

and

for every other possible match C for B, $v(A,B) < v(C,B)$.

This algorithm does not guarantee 100% success in the matching process but in most of the hallway scenes we have analyzed, we did attain 100% success.

To illustrate the sort of results obtained, the vertical lines detected in the stereo pair of Fig. 3.11 are shown in Fig. 5.6. There were 10 vertical lines detected in the left image and 15 in the right image, the different numbers owing to the differences in the viewpoints and photometric variations in the two images of the stereo pair. Clearly, any matching process must not form more than 10 pairs. Our algorithm did exactly that and formed a verifiably correct match for each of the lines shown in Fig. 5.7.

5.3 RECONSTRUCTING 3D SCENE LINES

From the pairs of vertical lines matched from the left and the right images, the next task is to generate the equations of the scene vertical lines that gave rise to the image vertical lines. In this section, we will show how that can be done.

First, in Section 5.3.1, we will show that in a fashion analogous to the association of a line of sight with each pixel in an image, we can associate a plane of sight with each line extracted from an image. Given the coordinates of a pixel, Eq. (3.20) gives us an equation for the line of sight that goes with that pixel. In a spirit similar to that of Eq. (3.20), given the equation of a line extracted from an image, we want to derive the equation of a plane which contains that line and the lens center -- by definition the plane of sight. Note that a plane of sight contains the lines of sight for all the pixels on an image line.

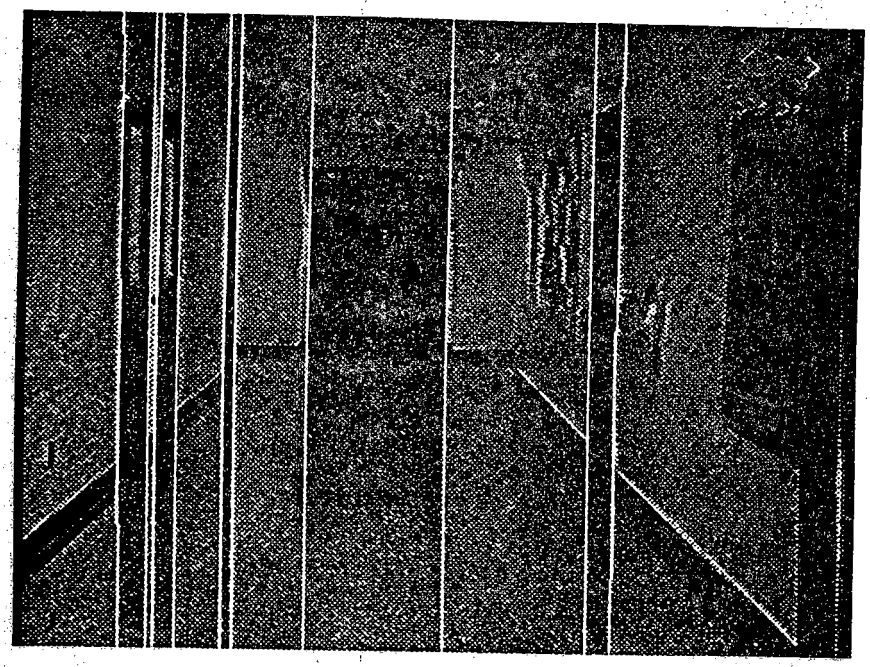
Then in Section 5.3.2, we will show how by intersecting the planes of sight corresponding to the two image lines of a matched pair, we can derive the equation of a scene line that gave rise to the image lines in the left and the right images.

5.3.1 Planes of Sight

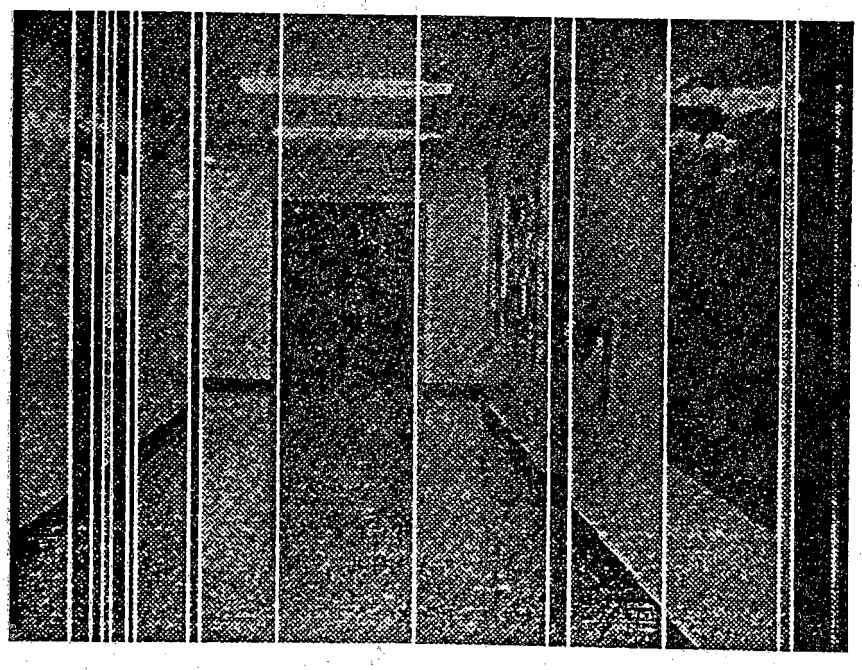
In order to derive the equation of the plane of sight associated with a line extracted from an image, we start with the slope-intercept equation of the line

$$u = m\,v + c \tag{5.2}$$

where m is the slope and c the x-intercept. Note that we have reverted back to image coordinate frame used prior to the Hough transformation step of Chapter 4, meaning the x-axis is directed along the scan lines of the image plane and the y-axis along the perpendicular to the scan lines. We will use (u,v) to denote the coordinates of a point in the image frame, as opposed to the indices i and j of Chapter 3, the reason being that we now allow the coordinates of an image point to be real numbers; note that an arbitrary point on a line corresponding to the estimated values of m and c may not fall on one of the



(a) Left



(b) Right

Figure 5.6: Lines detected in the left and right images.

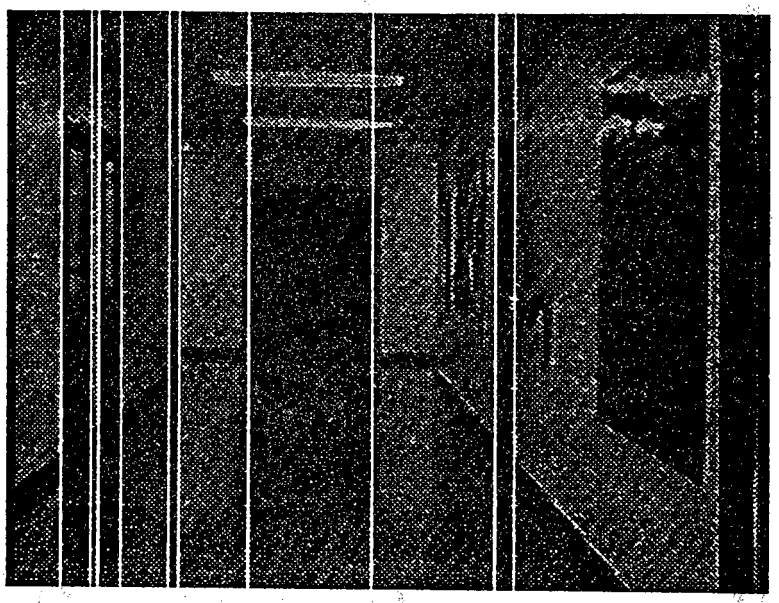


Figure 5.7: Lines correctly matched.

sampled pixels in the image. The world coordinates will continue to be designated by (x,y,z).

In contrast with Eq. (5.2), a line in an image may also be described by the following vector equation

$$\vec{P_I} = \lambda \vec{k} + \vec{U_0} \tag{5.3}$$

where $\vec{P_I}$ is a vector in the image coordinate frame to a particular point on the line, $\vec{k}$ a vector along the direction of the line, and U_0 a vector to a reference point on the line. The parameter λ is needed so that the length of the vector $\lambda\vec{k}$ is equal to the distance from the reference point $\vec{U_0}$ to the point $\vec{P_I}$ on the line. Clearly, the same line in the image frame can be described by many different values for $\vec{k}$ and $\vec{U_0}$. In particular, we may express them in terms of the known m and c as follows:

$$\vec{k} = (m,\ 1) \qquad \vec{U_0} = (c,\ 0) \tag{5.4}$$

In other words, we use the intercept with the x-axis as the reference point for the vector equation, and define the line direction vector $\vec{k}$ directly in terms of the slope m. Eq. (5.3) may also be written in the following form:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \lambda \begin{bmatrix} k_u \\ k_v \end{bmatrix} + \begin{bmatrix} u_0 \\ v_0 \end{bmatrix} \tag{5.5}$$

where (u,v) are the two components of the vector $\vec{P_I}$ to the image point, as they appear in Eq. (5.2), $(k_u,\ k_v)$ the two components of the line-direction vector $\vec{k}$ -- of course, in our case, k_u equals m and k_v 1 -- and, $(u_0,\ v_0)$ the two components of the vector to the reference point, with u_0 equal to c and v_0 equal to 0 in our case.

As mentioned before, Eq. (3.20) gives us the parametric equation, $\vec{P_W} = \mu\vec{D} + \vec{C}$, of the line of sight that corresponds to the pixel at location $\vec{P_I}$ in the image plane, with $\vec{D} = R^{-1}\vec{P_I}$; note that vector $\vec{C}$ corresponds to the location of the camera lens center (see Fig. 3.3). The matrix R is the upper left 3×3 submatrix of the calibration matrix T whose components are computed by the method described in Chapter 3. The vector $\vec{C}$ can be computed from the last column of the calibration matrix from Eq. (3.14). To derive the equation of a plane of sight, we need to combine Eq. (3.20) with Eq. (5.5). However, before we can do so, Eq. (5.5) must be expressed in homogeneous coordinates:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \lambda \begin{bmatrix} k_u \\ k_v \\ 0 \end{bmatrix} + \begin{bmatrix} u_0 \\ v_0 \\ 1 \end{bmatrix} \tag{5.6}$$

or, equivalently, as

$$\vec{P_I} = \lambda\vec{K} + \vec{U_0} \tag{5.7}$$

Combining Eqs. (3.20) and (5.7), we get the following equation for the plane of sight

$$\vec{P}_W = \mu\lambda R^{-1}\vec{K} + \mu R^{-1}\vec{U}_0 + \vec{C} \tag{5.8}$$

If we use the substitutions $\vec{P}_1 = R^{-1}\vec{K}$, $\vec{P}_2 = R^{-1}\vec{U}_0$ and $\lambda' = \mu\lambda$, we can write

$$\vec{P}_W = \lambda'\vec{P}_1 + \mu\vec{P}_2 + \vec{C} \tag{5.9}$$

which represents the parametric equations of a plane in 3D space. To present this equation in a more familiar form we could multiply both sides by

$$\vec{n} = \vec{P}_1 \times \vec{P}_2 \tag{5.10}$$

obtaining

$$\vec{n}\cdot\vec{P}_W = \vec{n}\cdot\vec{C} = p'_0 \tag{5.11}$$

If we normalize by dividing both sides by $|\vec{n}|$, we get

$$\hat{n}\cdot\vec{P}_W = p_0 \tag{5.12}$$

which is a succinct representation of a plane of sight. Eq. (5.11) tells us that the plane of sight contains the camera lens center, since $\vec{C}$ is the vector to the lens center in the world frame.

5.3.2 From Planes of Sight to 3D Scene Lines

Given a matched pair of vertical lines, one from the left image and the other from the right image, we can construct planes of sight for each image line by the procedure just described. By intersecting the two planes of sight, we can then derive the scene line that gave rise to the image lines in the two cameras.

In keeping with Eq. (5.12), let $\hat{n}_l\vec{P} = p_l$ and $\hat{n}_r\vec{P} = p_r$ be the equations of the planes of sight corresponding to the two vertical lines in a matched pair; the subscript l refers to the left image and the subscript r to the right image. The direction of the 3D scene line generated by the intersection of these planes of sight is given by

$$\vec{r} = \hat{n}_l \times \hat{n}_r \tag{5.13}$$

Now that we have the direction of the line, we need to constrain the location of the line. This we do by computing the intersection of the scene line with the floor defined by $z = 0$. This point on the floor, which will be designated by the symbol $\vec{P}_{W0}$, can be computed as the intersection of three planes, the two planes of sight and the $z = 0$ plane. In general, the vector to the intersection point of three planes that are at arbitrary orientations vis-a-vis one another is given by

$$\vec{P}_0 = \frac{p_1(\hat{n}_2 \times \hat{n}_3) + p_2(\hat{n}_3 \times \hat{n}_1) + p_3(\hat{n}_1 \times \hat{n}_2)}{\hat{n}_1\cdot(\hat{n}_2 \times \hat{n}_3)} \tag{5.14}$$

where we have assumed that the equations of the three planes are

$$\begin{aligned} \hat{n}_1 \cdot \vec{P} &= p_1 \\ \hat{n}_2 \cdot \vec{P} &= p_2 \\ \hat{n}_3 \cdot \vec{P} &= p_3 \end{aligned} \tag{5.15}$$

For our case, since one of the planes is given by $z = 0$, meaning that the plane passes through the world origin, we can set $p_3 = 0$. Therefore, the expression for the intersection of the 3D scene line and the floor reduces to

$$\vec{P}_{W0} = \frac{p_l (\hat{n}_r \times \hat{k}) + p_r (\hat{k} \times \hat{n}_l)}{\hat{n}_l \cdot (\hat{n}_r \times \hat{k})} \tag{5.16}$$

where $\hat{k}$ is the unit vector in the direction of the z-axis. Eqs. (5.13) and (5.16) define the 3D scene line with the help of the following parametric form

$$\vec{P}_W = \lambda \vec{r} + \vec{P}_{W0} \tag{5.17}$$

5.4 EXPERIMENTAL RESULTS

Following the procedure described in the preceding sections, we computed the equations of the 3D scene lines corresponding to the matched pairs of vertical lines from the stereo pair in Fig. 5.6. The x and y coordinates of the intersections of these scene lines with the floor are shown in Tables 5.1 and 5.2. [The robot was placed at the origin of the world frame for these experiments so that the robot frame was coincident with the world frame.] The computed x and y coordinates were compared with their ground truth values and two different errors determined. Table 5.1 shows the radial error, this is the error in the distance to the scene vertical line from the world origin. On the other hand, Table 5.2 shows the errors in each of the two coordinates of the intersections of the vertical lines with the floor. Note for the segment of the hallway shown in Fig. 5.6, the world x-axis is perpendicular to the walls on the two sides of the robot and the world y-axis along the walls on the floor (the z-axis being the vertical). Given that the y-axis is nearly parallel to the optic axes of the cameras, it is not surprising that the y-coordinate errors are larger.

If the errors between where the robot actually is and where it thinks it is are small enough, the computed intersections of the scene vertical lines with the floor can be used directly for self-location. In this case, the computed coordinates of such an intersection on the floor will be off somewhat from their true values, but small enough so that the true coordinates of such points can be used to eliminate the errors in the location of the robot. For this scheme to work, the robot must stop frequently for updating its location. When

longer hop lengths are desired, it becomes necessary to carry out a graph-theoretic comparison of the shapes made by the intersections of the vertical lines with the floor with the shapes extracted from a model of the hallways. Procedures for carrying out such comparisons have not yet been implemented are topics of current research in the lab.

Table 5.1: Radial errors for the lines of Fig. 5.7.

True coordinates		Radial errors	
X (m.)	Y (m.)	∇r (m.)	$\nabla r / r$ %
-1.34	8.69	0.06	0.73
-1.34	9.71	0.15	1.52
-1.34	10.08	0.02	0.25
-1.34	11.10	0.27	2.45
-1.34	15.53	0.48	3.07
-1.34	15.79	0.77	4.84
-0.81	18.62	0.73	3.94
0.97	18.62	2.03	10.90
1.48	9.97	0.42	4.23
1.48	8.79	0.33	3.70

Table 5.2: Coordinate errors for the lines of Fig. 5.7.

True coordinates		Coordinate errors			
X (m.)	Y (m.)	∇x (m.)	∇y (m.)	$\nabla x / r$ %	$\nabla y / r$ %
-1.34	8.69	0.00	0.06	0.04	0.73
-1.34	9.71	-0.02	0.15	-0.26	1.5
-1.34	10.08	0.01	-0.02	0.10	-0.22
-1.34	11.10	0.03	-0.27	0.30	-2.44
-1.34	15.53	0.06	0.47	0.39	3.05
-1.34	15.79	-0.05	0.76	-0.33	4.82
-0.81	18.62	-0.02	-0.73	-0.11	-3.94
0.97	18.62	0.16	2.03	0.84	10.89
1.48	9.97	-0.07	-0.42	-0.68	-4.17
1.48	8.79	0.03	0.32	0.38	3.68

REFERENCES

[1] G. Medioni and R. Nevatia, "Matching images using linear features", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-6, no. 6, pp. 675-685, 1984.

[2] B. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision", *Proc. Image Understanding Workshop*, pp. 121-130, 1981.

[3] Y. Shirai, *Three-Dimensional Computer Vision*, Springer-Verlag, 1987.

[4] G. Medioni and R. Nevatia, "Segment-Based Stereo Matching", *Computer Vision, Graphics, and Image Processing*, vol. 31, pp. 2-18, 1985.

[5] W Grimson, *From Images to Surfaces*, MIT Press, 1981.

[6] D. Marr and T. Poggio, "A Computational Theory of Human Stereo Vision", *Proc. of Royal Society of London*, B, vol. 204, pp. 301-328, 1979.

[7] A. C. Kak, "Depth perception for robots", *Handbook of Industrial Robotics*, pp. 272-319, John Wiley, New York, 1985.

relationship between the lines of sight corresponding to the pixels of the image and the world frame. For our illustration, the rendered expectation map is shown (a) of Fig. 6.1. It is important to realize that the camera image in (b) and the expectation map in (a) are not in registration. Also important is the fact that the misregistration between the two is not a simple translation or rotation or any combination of the two. In the rest of this discussion, the expectation image of (a) will also be referred to as the model image and the different geometric entities in a model image will sometimes be referred to as model labels. On the basis of the similarity between the model entities and the scene entities, the model labels can be given to the geometric entities extracted from (b) and a confidence value associated with the label assignment.

PSEIKI first applies a preprocessor to the image of Fig. 6.1b. The preprocessor extracts edges and regions from the image. Thick edges are thinned to make them one-pixel wide and any small breaks between the edges that are nearly collinear are repaired. The details on preprocessing can be found in [2]. The output of the preprocessor for the image of (b) is shown in (c) in Fig. 6.1. PSEIKI's main job is make a comparison of the image of (c) with the model image of (a) and, via such a comparison, associate the model labels with the entities in (c). As we will show in this chapter, such associations can then be used to figure out the exact location of the robot, in other words for zeroing out the uncertainties associated with the robot position and orientation.

As was mentioned before, PSEIKI compares the images in (a) and (c) of Fig. 6.1 at different levels of geometric abstractions. The edge-level comparison yields the labeling of the scene edges in (c) with the model edge labels from (a), each such label assignment carrying a belief value. The face-level comparison yields face-level model labels for the regions formed by the closed edge contours in (c). Again, a belief value is associated with each model face label for a region from (c); however, this belief is now influenced by the model labels of the edges composing the region and the beliefs associated with the edge-level labels involved. If after PSEIKI has stopped processing the data, we identify the most-believed highest-level abstraction from (c), and then display the edges corresponding to this abstraction, we get the output shown in (d). In other words, we have the highest confidence in the model labels for the scene data edges shown in (d). These data edges and their model labels are then used for calculating a precise fix on the location of the robot by using the methods of this chapter. But, first, a few words are in order on the overall reasoning architecture of PSEIKI.

PSEIKI's architecture is shown in Fig. 6.2. The system has been implemented in OPS83 as a two-panel five-level blackboard [4]. The left panel, called the *model panel*, holds the abstraction hierarchy for the expected scene. The right panel, called the *data panel*, holds the abstraction hierarchies built from the scene data by taking cues from the supplied model abstractions in the left panel. As shown, each panel consists of different levels, each level corresponding to a different level of geometric abstraction. The lower

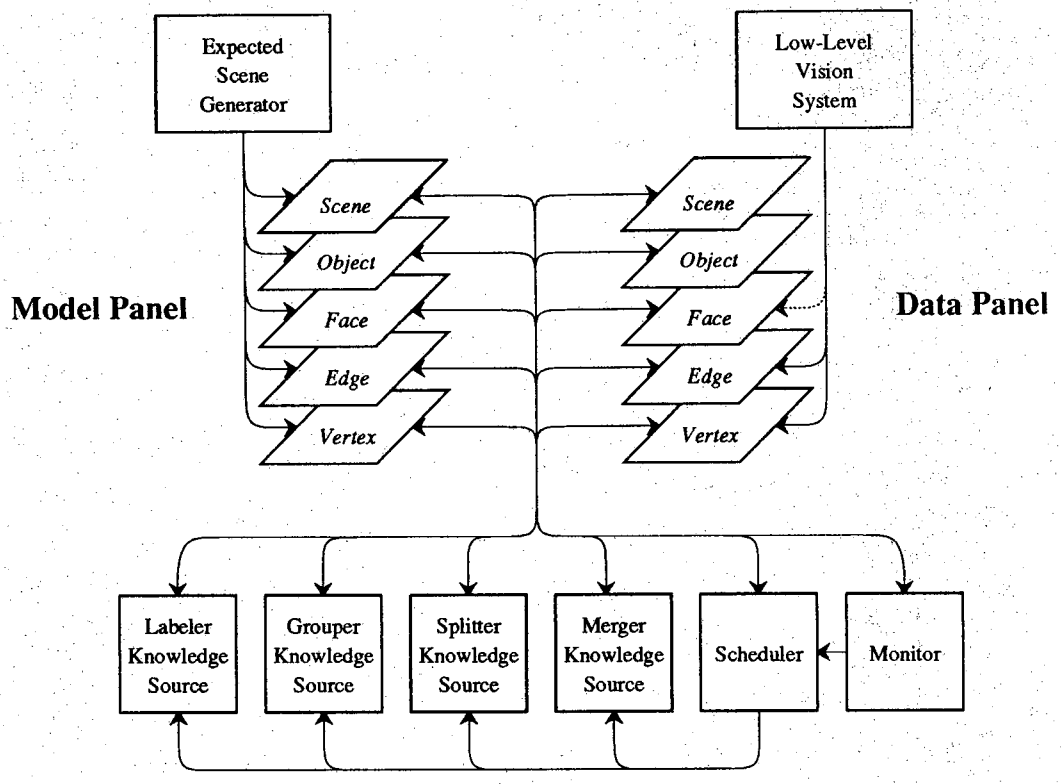


Figure 6.2: PSEIKI's architecture.

levels of the *data-panel* are supplied with the symbolic output of the image preprocessor, such as represented by (c) of Fig. 6.1.

As was mentioned before, PSEIKI associates model labels with scene entities at different levels of abstraction; each such label assignment is given a belief value based on the geometric similarity of the model entity represented by the label and the scene entity. While a direct geometric comparison is sufficient for the calculation of belief values at the edge level, at the face and higher levels the process of belief value calculation is in actuality more complex since, in addition to the geometric similarity of, say, a data face with a model face, we must also consider the beliefs associated with their children, the edges. In other words, PSEIKI must use a calculus of beliefs for accumulating beliefs for abstractions at face and higher levels; PSEIKI uses the Dempster-Shafer theory of evidence [5] for that purpose. A particular advantage of this formalism is that it allows the system to express ignorance about assigning a model label to a data entity; this is useful when the data entity does not match any model element to a sufficiently high degree. To overcome the exponential complexity usually associated with the Dempster-Shafer formalism, a computationally efficient variation of Dempster's rule is used to combine evidence [4].

PSEIKI has four main knowledge sources (KSs) that it uses to establish correspondences between the model entities and the scene data entities: *Labeler*, *Grouper*, *Splitter*, and *Merger*. Basic to the operation of the blackboard is the notion of a model label for a scene data entity. Actually, there is a set of competing labels created for the scene entities -- this set is called the *frame of discernment* (FOD) for the scene entity -- but the most believed of the members of the FOD is called the *label* of the scene entity. As different geometrical and relational constraints are invoked at different levels of abstractions, beliefs are continually accumulated for all the members of the FOD; if through this accumulation the belief in the current label for a scene entity becomes less than the belief in some other member of the FOD, then the label of the scene entity gets changed to that member of the FOD.

The Grouper KS determines which scene entity at a given level of the hierarchy should be grouped together to form an entity at a higher level of abstraction. As an example, which edges should be grouped together to form a face, etc. Grouping proceeds in a data-driven fashion in response to goals that call for the establishment of nodes on the right panel corresponding to the model nodes in the left panel. The grouping goals are generated by the Scheduler initially, since one of the initial jobs given to the Scheduler is to scan the model panel top to bottom and generate goals for the creation of a certain minimum number of competing scene nodes corresponding to each model node. In response to a goal for the creation of, say, a scene face corresponding to a model face, the Scheduler looks for an edge that has the strongest attachment, on the basis of belief values, to any of the edges of the model face in question. This scene data edge then

serves as a seed for initiating a grouping by invoking geometrical considerations. The Labeler KS has three different functions assigned to it: 1) determination of initial model labels for the scene edges and faces based purely on spatial proximity and geometric similarity; 2) belief revision for face and higher level scene nodes on the basis of relational considerations amongst the children nodes; and 3) propagation of beliefs up the hierarchy.

The Merger KS also groups elements; however, its job is to merge multiple elements at a given level and retain the grouped information at the same level. As an example, if two scene faces have the same model label and there exists a long enough common boundary between the two faces, the Merger will merge the two faces into a single face and subsequent computations. The action of the Splitter KS is opposite to that of the Merger; it splits a single element on the blackboard into multiple smaller elements. For example, a T junction may be split up into two or three separate edges.

The overall flow of control is controlled by the Monitor and the Scheduler, acting in concert. The Monitor uses OPS demons to run in the background, its task being to watch out for the data conditions that are needed for triggering the various KS's. For example, if there is a scene edge without a parent, it is the Monitor's job to become aware of that fact and synthesize a knowledge source activation record (KSAR) that is a record of the identity of the edge element and the KS that can be triggered by that element. Initially, when the KSAR's are first created, they are marked as pending. When no KS is active, the Scheduler examines all the pending KSAR's and selects one according to prespecified policies. For example, the status of a KSAR that tries to invoke the Merger or the Splitter KS is immediately changed to active. It seeks intuitively reasonable to fire these KS's first because they seek to correct any misformed groups.

6.2 EXPECTED SCENE GENERATION

The TWIN boundary-representation (B-rep) solid modeling system is used to generate PSEIKI's expected-scene data. TWIN is a library of C language subroutines; the library contains routines for generating a number of primitive objects including parallelepipeds, wedges, cylinders, cones, toruses, spheres, fillets, elliptical cones, and ellipsoids. The library also contains routines for performing regularized Boolean operations, such as the union, intersection, and difference operations, on solid objects. Complex solid objects are created by combining sub-objects with these operators. For our work, a solid model of the building's corridors is generated off-line and is used to generate the robot's expectation map, such as the one shown in Fig. 6.1a. The process of expected scene generation may be conceived of along the following lines: As the robot travels physically down a hallway, in the computer memory it travels down a solid model --

within of course the accuracy afforded by the odometry. Every once a while, the robot compares what its one eye is seeing with its 'mental' picture of what it should be seeing and this comparison leads to the robot becoming aware of its exact position.

The expected scene image of Fig. 6.1a is actually a result of a two step procedure. First, given the odometry-supplied position and orientation of the robot, a scan-line algorithm is used on the TWIN model of the hallways to render an expected scene image. The rendered image at this point is a gray level image, the gray level at each pixel corresponding not to any photometric information but to the ID number of the hallway surface visible at that pixel. Next, this image is processed with the same preprocessor that is used on the scene image, except that now we have a 'perfect' image in the sense that each region has an absolutely constant gray level. The preprocessor then produces the kind of image shown in Fig. 6.1a.

6.3 COMPUTING POSITION AND ORIENTATION PARAMETERS

Corresponding to the most-believed overall scene interpretation, the output provided by PSEIKI consists of a set of image-edge to hallway-line correspondences and the associated belief values. While the image edges and the corresponding hallway lines are all described by the coordinates of their vertices, there will almost never be a direct correspondence between the image vertices and the hallway vertices, the reason being the fragmentation of the image edges during segmentation.

Given these image-edge to hallway-line correspondences, the task faced by the Navigator is to determine the location and the orientation of the robot.

Since the floor is flat, the mobile robot has only three degrees of freedom in our experiments, namely, the location (X, Y) of the center of its base and its orientation (θ). The calculation of these two items will now be addressed in the next two subsections.

6.3.1 Orientation Calculation

In this section, we will show that sufficient information for computing the orientation of the robot consists of just one image edge together with its hallway correspondent, provided the hallway line is not exactly vertical.

Let (v_x, v_y, v_z) be the components, in the *robot* frame of reference, of a unit vector along the 3D scene edge whose image is being used for orientation calculation. Now, as was done in Section 5.3.1, we may associate a plane of sight with the image edge under consideration; the plane of sight passes through the image edge and the camera lens

center. The equation of this plane of sight, derived from the slope-intercept parameters of the image edge and the camera calibration parameters, is given by Eq. (5.12) in the robot coordinate frame; in this equation $\hat{n} = (n_x, n_y, n_z)$ is the normal to the plane of sight. It is clear that both the actual scene edge and its image must lie in the same plane of sight, since, by definition, the plane of sight contains the lines of sight for all the pixels on the image edge. From this observation follows that $\hat{n}$ must be perpendicular to the direction vector of the scene edge

$$n_x v_x + n_y v_y + n_z v_z = 0 \tag{6.1}$$

Note again that v_x, v_y and v_z are the three components of the direction of the actual scene edge but in the robot coordinate frame. In the *world* coordinate frame, let the components of the direction unit vector for the same scene edge be given by (V_x, V_y, V_z). Since the robot is always on a flat floor, the z-axis of the robot coordinate frame is always parallel to the z-axis of the world coordinate frame and there is no z-direction displacement between the two origins. Therefore, it must be the case that v_z and V_z are the same. This observation, when used in Eq. (6.1), leads to the first of the following two equations, the second equation follows from a similar substitution in the equation that says that the magnitude of the (v_x, v_y, v_z) vector is unity.

$$n_x v_x + n_y v_y = -n_z V_z \tag{6.2}$$

$$v_x^2 + v_y^2 = 1 - V_z^2 \tag{6.3}$$

So we have two equations for the two unknowns v_x and v_y, but only when the hallway line is not vertical; when the hallway line is vertical, V_z will equal unity and the second equation will become non-existent. As was mentioned before, the quantities n_x, n_y, and n_z are known from Eq. (5.12) and are calculated by fitting a plane of sight, in the robot coordinate frame, to the image edge being used for orientation calculation.

The conditions under which the pair of equations, (6.2) and (6.3), has one or more solutions can be inferred by examining the equations in the (v_x, v_y) space. In this space, Eqs. (6.2) and (6.3) represent a line and a circle, respectively, as shown in Fig. 6.3. Whether there are two, one, or no solutions depends on whether the line intersects, is a tangent, or is exterior to the circle, respectively. In our hallways, practically all the lines are either entirely vertical or entirely horizontal; in other words, all the hallway lines corresponding to the edges that can be seen in (b) and (c) of Fig. 6.1 are either vertical or horizontal in the world coordinate frame (even though that may not appear to be the case from the scene expectation map of Fig. 6.1a). Since, for reasons given above, the vertical lines in the hallway can not be used for determining the orientation of the robot, we must use the horizontal lines. Therefore, for the scene edge that is being used for figuring out the orientation of the robot it must be the case that $V_z = 0$. Note that $V_z = 0$ condition means that in the (v_x, v_y) space space, the line represented by Eq. (6.2) passes through

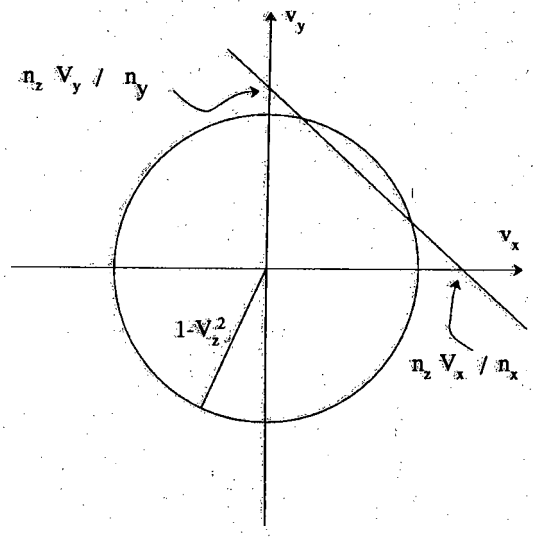


Figure 6.3: Equations 6.2 and 6.3 represented in (v_x, v_y) space.

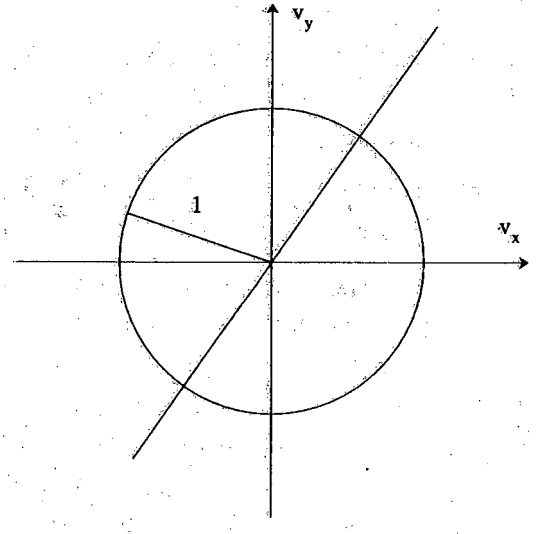


Figure 6.4: Equations 6.2 and 6.3 represented in (v_x, v_y) space when only horizontal lines in 3D space are possible.

the origin, as shown in Fig. 6.4. This guarantees that there will be two solutions for the unknowns v_x and v_y.

So we now know how to compute the projections v_x and v_y of the hallway line being used for orientation calculation on the x and y axes of the robot coordinate frame. Since from the correspondence established by PSEIKI, we also know the actual model identity of the hallway line, the quantities V_x and V_y are also known, these being the projection of the same hallway line on the world x and y axes. The orientation of the robot can now be easily computed from the relationship between (v_x, v_y) and (V_x, V_y). Recall that while the robot coordinate frame rotates and translates vis-a-vis the world coordinate frame, the x,y-planes of the two coordinate frames must always stay coplanar. It therefore follows that the relationship between (v_x, v_y) and (V_x, V_y) is

$$\begin{aligned} v_x &= V_x \cos\theta + V_y \sin\theta \\ v_y &= -V_x \sin\theta + V_y \cos\theta \end{aligned} \tag{6.4}$$

where θ is the angle between the y axes of the two coordinate frames. The reader might wonder about the validity of Eq. (6.4) since it represents the transformation between two 2-D coordinate frames when one undergoes a rotation with respect to the other, without there being any translational displacement between the two; while in our case the robot coordinate frame would have undergone translation in addition to, of course, the rotation. The reason that Eq. (6.4) is valid notwithstanding the translation has to do with the fact that (v_x, v_y, v_z) and (V_x, V_y, V_z) both are direction vectors of unit magnitude, rooted at the origins of their respective coordinate frames. What's being said is that while Eq. (6.4) would not, in general, be a valid transformation for computing the robot frame coordinates of a point whose position is specified in the world coordinate frame or vice versa, it is a perfectly valid transformation for the direction vectors.

Eqs. (6.2) and (6.3) give us two solutions, (v_x, v_y) and $(-v_x, -v_y)$, that in turn result in two solutions for θ that differ by 180°. Assuming that the robot orientation is approximately known to start with, choosing the correct solution is not difficult, but if this information is not available an additional 2D to 3D correspondence can resolve the ambiguity.

6.3.2 Calculating the World Location of the Origin of the Robot Coordinate Frame

We will start out by showing that from a single image edge corresponding to a horizontal line feature in the hallway, it is possible to calculate the perpendicular distance of the origin of the robot coordinate frame from the hallway line. However, knowing the perpendicular distance of the robot-base center from a hallway feature constrains only one of the two degrees of freedom that characterize the translation of the robot on the

floor. Therefore, if one were to use this procedure, such perpendicular distances would have to be found from at least two non-parallel horizontal lines of the hallway.

Next, we will derive a more easily implementable method that also uses two non-parallel hallway lines, and their image edges as supplied by PSEIKI, to directly yield the world coordinates of the origin of the robot coordinate frame. This method is predicated on the assumption that the orientation of the robot has already been determined.

Finally, we will present yet another method for the computation of the displacement between the origin of the robot coordinate frame and the world coordinate frame; this method uses a single image point and its corresponding hallway point. Although straightforward to implement, this method can only be used if two image edges can be found that either meet or intersect at a point under the condition that the hallway lines corresponding to the two image edges also meet or intersect at a hallway point that corresponds to the image point given rise to by the image edges. This requirement makes this method less useful.

Much discussion in this section will make frequent references to two coordinate frames: *the world coordinate frame*, which stays stationary and in which the 3D model of the hallways exists, and the *robot coordinate frame*, which always translates and rotates with the robot. Both of these are shown in Fig. 6.5, where the axes denoted by (x_W, y_W, z_W) represent the world coordinate frame and those denoted by (x_R, y_R, z_R) represent the robot coordinate frame. As shown in the figure, the image edge *ab* presumably corresponds to the hallway line *AB* and this fact has supposedly been discovered by PSEIKI. The point *C* represents the camera lens center; it is defined by the vector $\vec{c}$ in the robot coordinate frame. The vector $\vec{D}_{tran}$ denotes the translation of the robot coordinate frame with respect to the the world coordinate frame. Note that $\vec{D}_{tran}$ contains no information regarding the orientation of the robot -- recall that the orientation is the angle between the y-axis of the robot coordinate frame and the y-axis of the world coordinate frame. In this section, our goal is to calculate $\vec{D}_{tran}$, assuming the orientation of the robot coordinate frame has already been calculated by the method of Section 6.3.1.

Note that for the image edge *ab* shown in Fig. 6.5, the plane of sight is formed by joining the camera lens center with any two points on the edge *ab*. This plane of sight, marked π, has normal $\hat{n}$ and its equation, derived by the method of Section 5.3.1 from the image edge parameters and the calibration matrix, is defined in the robot coordinate frame. In other words, the Eq. (5.12) for the plane of sight π exists in the robot coordinate frame. Our mission is to use the equation of π in the robot coordinate frame and the equation of the line *AB* in the world coordinate frame for the calculation of $\vec{D}_{tran}$.

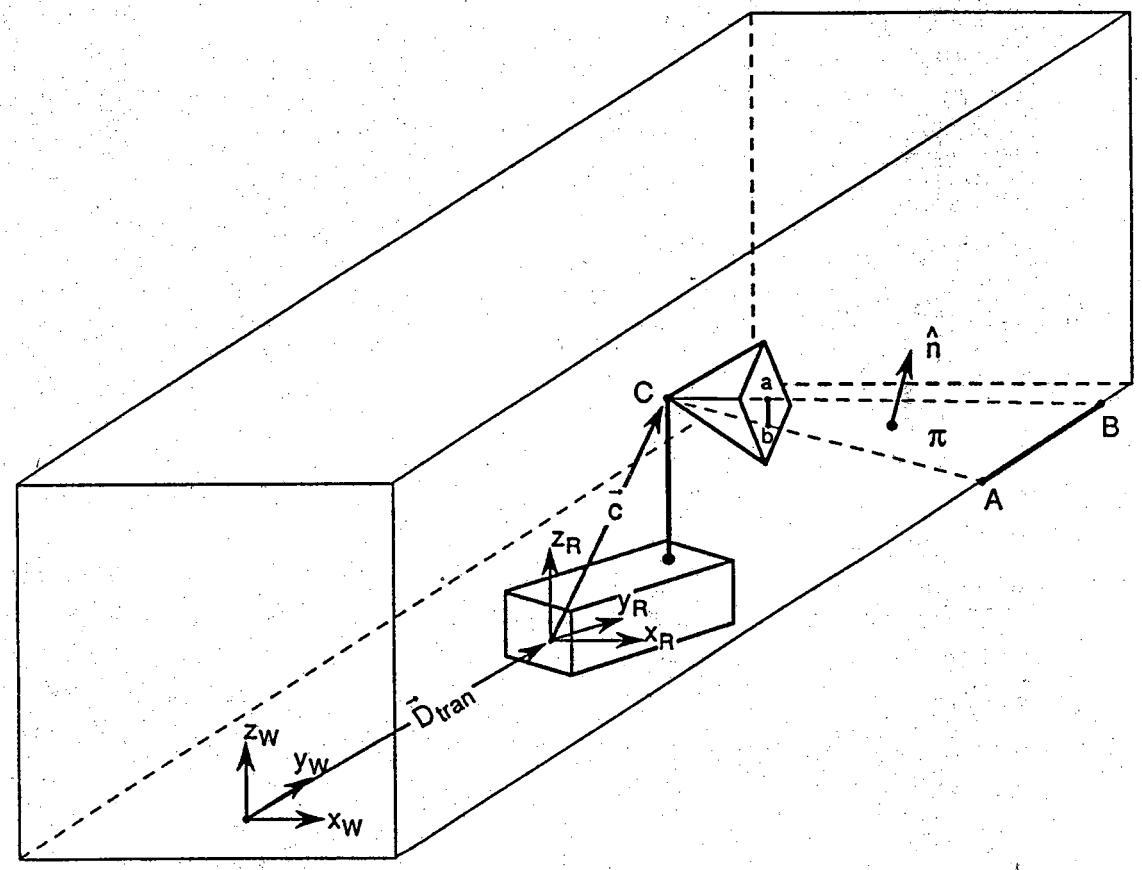


Figure 6.5: Shown are the world coordinate frame and the robot coordinate frame. The hallway model exists in the world coordinate frame, whereas the equation of the plane of sight corresponding to the image edge *ab* is defined in the robot coordinate frame.

SINGLE LINE CORRESPONDENCE METHOD:

We will now show how we might compute the perpendicular distance of the origin of the robot coordinate frame from a horizontal line feature in the hallway. For the sake of discussion, assume that we are using the image edge *ab*, shown in Fig. 6.5 for this purpose, and its hallway correspondent *AB*, the correspondence having been discovered by PSEIKI.

The applicable formula will be derived from Fig. 6.6; the plane of this figure is perpendicular to the floor, perpendicular to the horizontal line *AB* of Fig. 6.5, and passes through the camera lens center, which is marked *C* in both Figs. 6.5 and 6.6. Since the plane of the figure in Fig. 6.6 is perpendicular to the hallway line, the line *AB* becomes a single point in the figure -- this point is marked *L*. The line marked *P* is the intersection of the plane of sight π of Fig. 6.5, which corresponds to the image edge *ab*, with the plane of the figure. The parameter H is the height of the camera lens center above the floor and h the height of the horizontal line above the plane of the floor. Although $h = 0$ in Fig. 6.5, we retain this parameter for the derivation here since our results can be used for a horizontal line at any height.

Also, shown in Fig. 6.6 is the normal $\hat{n}$ to the plane of sight; this normal is same as $\hat{n}$ in Fig. 6.5. Therefore, $\hat{n}$ in Fig. 6.5 is a known quantity. From the figure, the angle ϕ shown there is

$$\phi = \pi - \cos^{-1}(\hat{n} \cdot \hat{k}) \tag{6.5}$$

where $\hat{k}$ is the unit vector along the z-axis. Therefore, from our knowledge of $\hat{n}$ we can calculate the angle ϕ. From the figure, the perpendicular distance d to the line *AB*, the line being represented by the point *L* in Fig. 6.6, is given by

$$d = (H - h)\tan\phi \tag{6.6}$$

Practically all the horizontal lines in our hallways are parallel to either the world x_W-axis or the world y_W-axis. If one of these lines is used for the calculations, the perpendicular distance d will be parallel to the y_W-axis, if the hallway line is parallel to the x_W-axis; d will be parallel to the x_W-axis, if the hallway line is parallel to the y_W-axis. This fact can be expressed succinctly by

$$Q_{robot} = Q_{line} \pm d \tag{6.7}$$

where Q_{robot} and Q_{line} stand, respectively, for the x_W components of $\vec{D}_{tran}$ and the line if the hallway line is parallel to the world y_W-axis, since in that case the equation of the hallway line is $Q_{line} = constant$. On the other hand, if the hallway line is parallel to the world x_W-axis, Q_{robot} and Q_{line} stand for the y_W components of $\vec{D}_{tran}$ and the line, respectively. Whether there should be a plus or a minus sign in the right hand side of Eq. (6.7)

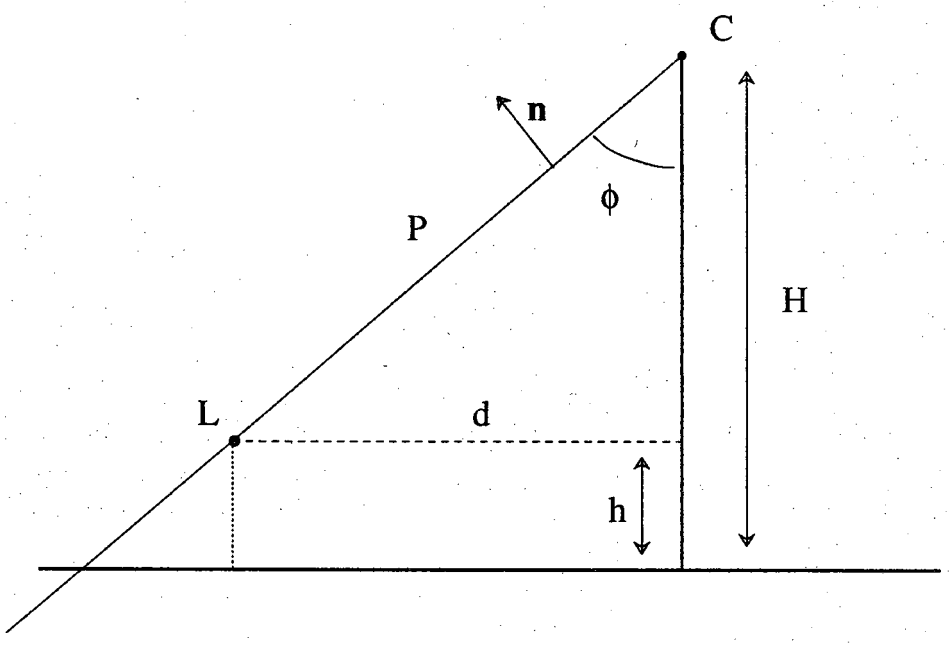


Figure 6.6: Determination of the perpendicular distance to a horizontal line.

can be inferred from the expected value of the coordinate Q_{robot}.

TWO LINE CORRESPONDENCES METHOD:

We will assume that the orientation of the robot has already been determined by the method of Section 6.3.1 and all that remains is to find the x_W and y_W components of the translation vector $\vec{D}_{tran}$ shown in Fig. 6.5. Since the orientation is already known, we now rotate the robot coordinate frame, not physically but only in the computer program, so that all its axes become parallel to the world axes. In other words, we now create in the computer program the situation depicted in Fig. 6.7, where the robot is shown pointing straight ahead. As a result of this rotation, $\vec{D}_{tran}$ does not change.

Clearly, the equation of the plane of sight formed by the image edge *ab* will be different in the robot coordinate frame of Fig. 6.7, compared to what it was for the robot coordinate frame of Fig. 6.5. We have used the symbol π' to represent the plane of sight in Fig. 6.7. The equation of π' can be obtained by the same technique that is given in Section 5.3.1 except that now we must apply a rotation transform to the vectors $\vec{P}_1$, $\vec{P}_2$, and $\vec{C}$ of Eq. (5.9), the rotation corresponding to the angle θ calculated in Section 6.3.1. For the derivation to follow, we only need to know the value of $\hat{n}$, the normal to the plane π in Fig. 6.5, in the robot coordinate frame of Fig. 6.7. We will denote this transformed value of $\hat{n}$ by $\hat{n}'$. The value of $\hat{n}'$ can be easily obtained from $\hat{n}$ by applying to $\hat{n}$ a rotational transform matrix corresponding the angle θ. We may then write the following equation for π':

$$\hat{n}' \cdot \vec{p} = K \tag{6.8}$$

where $\vec{p}$ is the position vector to any arbitrary point with respect to the origin of the robot coordinate frame of Fig. 6.7. The constant K can be easily determined by realizing the plane π' must pass through the camera lens center whose location is given by the vector $\vec{c}$. Therefore, it must be the case that

$$\hat{n}' \cdot \vec{c} = K \tag{6.9}$$

Note that $\vec{c}$ is known through camera calibration. We can therefore write Eq. (6.8) as

$$\hat{n}' \cdot \vec{p} = \hat{n}' \cdot \vec{c} \tag{6.10}$$

Equivalently, we have for the equation of π'

$$n'_x p_x + n'_y p_y + n'_z p_z = n'_x c_x + n'_y c_y + n'_z c_z \tag{6.11}$$

Now let's consider an imaginary plane, denoted by π'', parallel to the plane π' but passing through the origin of the world coordinate frame. Let $\hat{n}''$ be the surface normal of this new plane. Since, in Fig. 6.7, the axes of the world and the robot coordinate frames

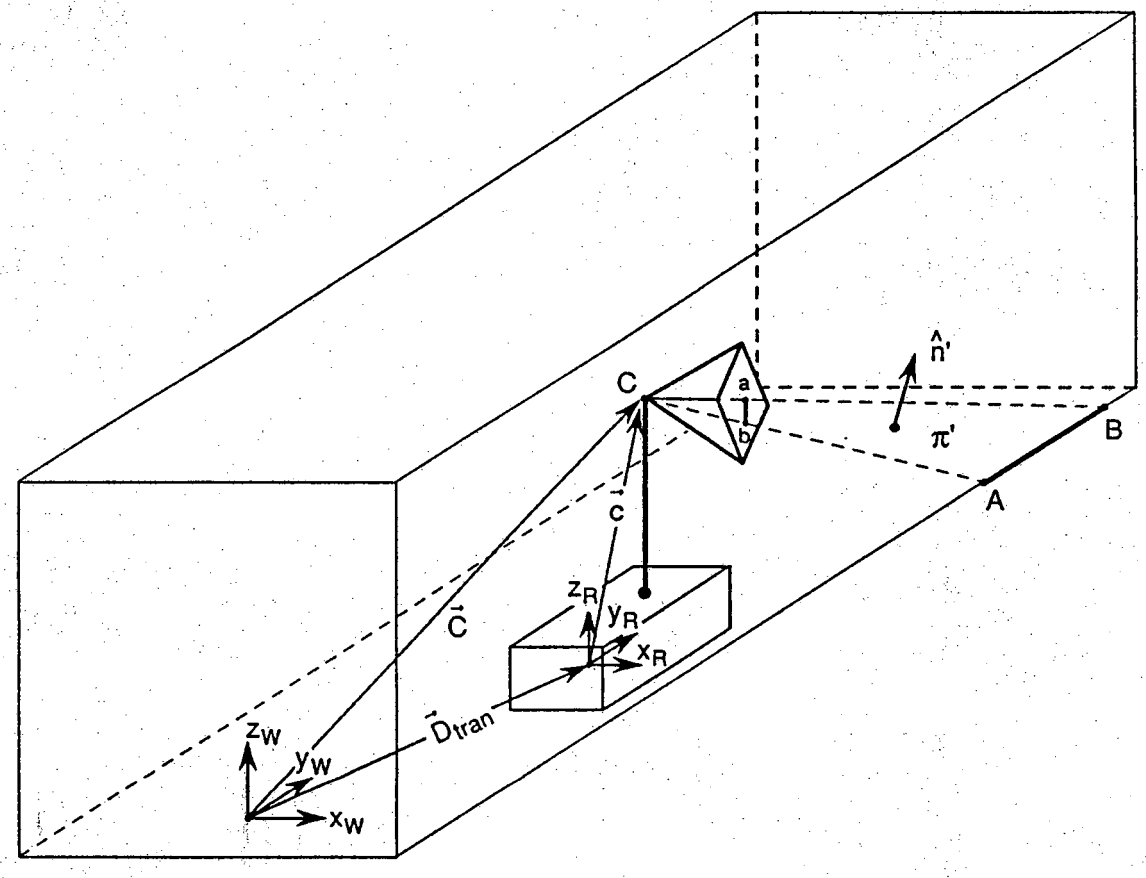


Figure 6.7: Same as Fig. 6.5, except that the robot coordinate frame has now been rotated about its origin through the angle θ calculated by the method of Section 6.3.1. Note that no physical rotation of the robot is called for, the rotation is carried out in software to facilitate the calculation of the translation vector $\vec{D}_{tran}$.

are parallel, we have

$$\begin{aligned} n''_x &= n'_x \\ n''_y &= n'_y \\ n''_z &= n'_z \end{aligned} \tag{6.12}$$

Since the plane π'' passes through the origin, its equation is

$$n''_x P_x + n''_y P_y + n''_z P_z = 0 \tag{6.13}$$

where (P_x, P_y, P_z) are the world coordinates of a point on π''.

The distance d between the planes π' and π'' can be computed by taking the projection, along the direction of the normal vector $\hat{n}'$, of any vector from any point on π'' to any point on π'. The origin of the world coordinate frame is on π'', therefore for any vector $\vec{P}_0$ from the origin of the world coordinate system to any point on π', we can write

$$d = n'_x P_{x0} + n'_y P_{y0} + n'_z P_{z0} \tag{6.14}$$

where (P_{x0}, P_{y0}, P_{z0}) are the world coordinates of any point on π'. We are particularly interested in two such points: the camera lens center, whose location is given by $\vec{c}$ in the robot coordinate frame and by $\vec{C}$ in the world frame, and any point (P_{x1}, P_{y1}, P_{z1}) on the hallway line AB in Fig. 6.7. Therefore we can write

$$n'_x C_x + n'_y C_y + n'_z C_z = d = n'_x P_{x1} + n'_y P_{y1} + n'_z P_{z1} \tag{6.15}$$

Note that the coordinates (C_x, C_y, C_z) are defined in the world coordinate frame; these coordinates are unknown at this time, since camera calibration, being carried out in the robot coordinate frame, only yields (c_x, c_y, c_z). On the other hand, the world coordinates (P_{x1}, P_{y1}, P_{z1}) of any point on the hallway line are known. Since $C_z = c_z$, we can rewrite (6.15) as

$$n'_x C_x + n'_y C_y = G \tag{6.16}$$

with $G = n'_x P_{x1} + n'_y P_{y1} + n'_z P_{z1} - n'_z c_z$. So we have one equation in Eq. (6.16) for the two unknowns C_x and C_y. A similar equation may be obtained from a second PSEIKI-supplied correspondence between some other image edge and a hallway line, giving us two equation for the two unknowns. We may now determine the components of the translation vector $\vec{D}_{tran}$ from

$$\begin{aligned} D_{tran,x} &= C_x - c_x \\ D_{tran,y} &= C_y - c_y \end{aligned} \tag{6.17}$$

Note that for a unique solution to exist for two equations of the form shown in Eq. (6.16), the two hallway lines must satisfy the conditions: 1) they must not be parallel to each other; and 2) the planes of sight must not be horizontal for either of the lines. If any of

these conditions is violated, the following determinant

$$\det_{1,2} = \begin{vmatrix} n_{x1} & n_{y1} \\ n_{x2} & n_{y2} \end{vmatrix} \tag{6.18}$$

would become zero and the two equations would cease to possess a solution.

POINT CORRESPONDENCE METHOD:

We will now present another method for the calculation of the displacement vector $\vec{D}_{tran}$ shown in Fig. 6.5. This method is based on the assumption that we can identify a single image point formed by the meeting or the intersection of two image edges such that the corresponding hallway lines also meet or intersect at a point that corresponds to the image point. Another assumption is that the orientation of the robot coordinate frame has already been computed by, for example, the method of Section 6.3.1. So, for the sake of computation, we may assume that the robot is turned around its origin through the rotation angle and its axes aligned with the world axes, as shown in Fig. 6.8.

Now, prior to the rotation of the robot coordinate frame, meaning in the robot coordinate frame shown in Fig. 6.5, let the equation of the line of sight to an image point corresponding to the world point P shown in Fig. 6.8 be $\vec{p} = \lambda \hat{v} + \vec{c}$. This equation, in principle the same as Eq. (3.20), is a parametric equation of the line of sight, the parameter being λ, $\hat{v}$ is the direction of the line formed by joining the camera lens center and the pixel p shown in Fig. 6.8, and c the camera lens center. As discussed in Section 3.2.1, this equation may be obtained by employing the submatrix D of the calibration matrix T.

When the robot coordinate frame is aligned as shown in Fig. 6.8, the equation of the line of sight to the pixel corresponding to the hallway point P will change. Let the direction of this new line of sight be $\hat{v}'$; $\hat{v}'$ is easily obtained by multiplying $\hat{v}$ by a rotational transform matrix corresponding to the planar rotation through angle θ calculated in Section 6.3.1. So, we can say that the equation of the line of sight to hallway point P in the robot coordinate frame of Fig. 6.8 can be written as $\vec{p} = \lambda \hat{v}' + \vec{c}$, where, for economy in notation, we have continued to use $\vec{p}$ for an arbitrary point on the line of sight and λ for the position parameter associated with this new line; the point $\vec{c}$ is the same as before since the camera lens center does not move with respect to the robot coordinate frame.

Given that the origin of the robot coordinate frame must always be in the (x_W, y_W) plane of the world frame and from the fact that the z-axes of the two frames are always parallel, it follows that for any arbitrary point P in the hallway, its z_W coordinate must be equal to its z_R coordinate. Let $\vec{p}_P$ be the robot coordinate frame vector to the world frame point P shown in Fig. 6.8. Since $\vec{p}_P$ must be a point on the line of sight, the z_R-coordinate of $\vec{p}_P$ must be given by $\lambda v'_z - c_z$, which is the same as $\lambda v'_z - C_z$, where C_z is the height

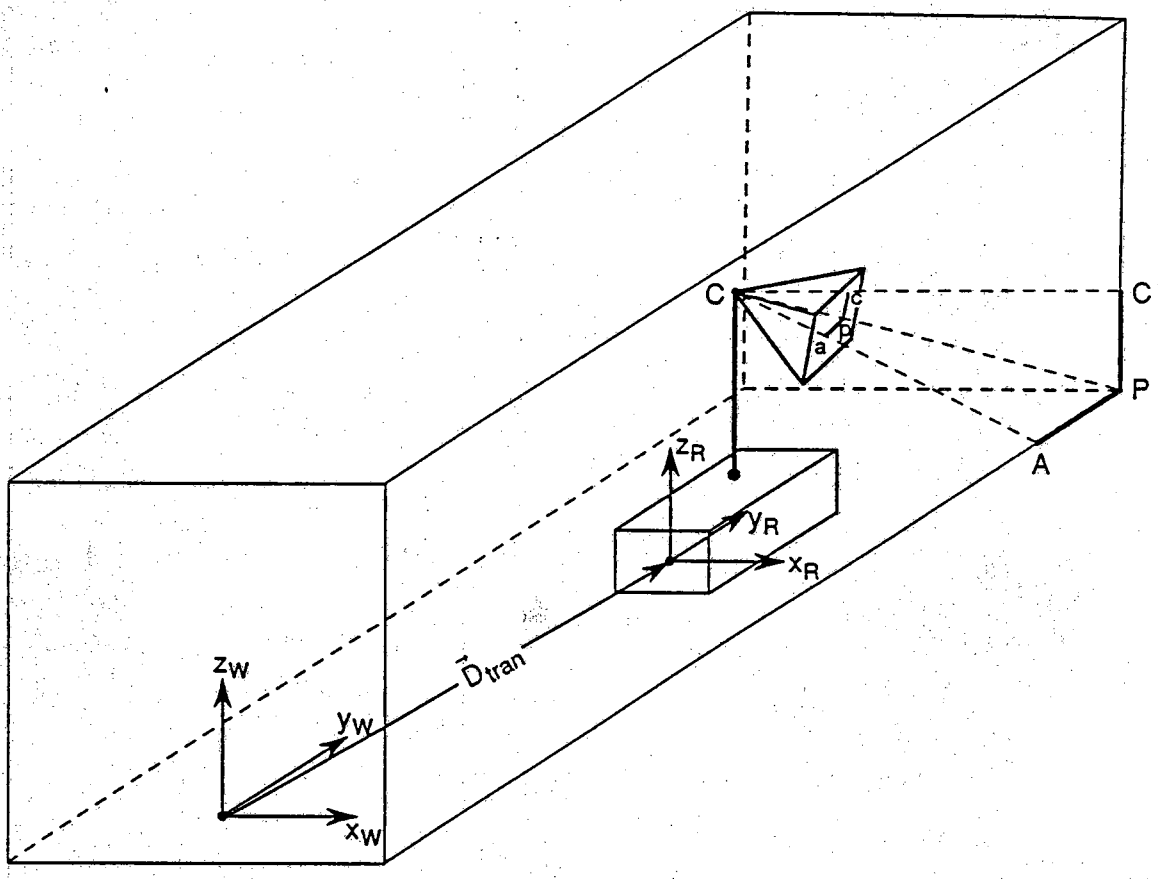


Figure 6.8: Same as Fig. 6.5, except that now the camera image contains a point corresponding to the intersection of two non-parallel hallway lines. Again, as in Fig. 6.7, for the sake of deriving a procedure for computing $\vec{D}_{tran}$, we show the robot coordinate frame axes aligned with the world coordinate frame axes.

of the camera lens center in the world coordinate frame, for some particular value of λ. If we assume that the z_W-coordinate of the hallway point P is P_z, then it must follow from the equality of the two z-coordinates that

$$\lambda_p = \frac{P_z - C_z}{v'_z}$$

where λ_p is the value of the parameter λ at the hallway point P on the line of sight. Now that we have fixed the value of the parameter λ, we can write down the following expressions for the robot frame coordinates of the hallway point P:

$$\begin{aligned} p_x &= \lambda_p v'_x + c_x \\ p_y &= \lambda_p v'_y + c_y \\ p_z &= P_z \end{aligned} \tag{6.19}$$

Computing the two components of $\vec{D}_{tran}$ is now straightforward. They are given by

$$\begin{aligned} D_{tran,x} &= P_x - p_x \\ D_{tran,y} &= P_y - p_y \end{aligned} \tag{6.20}$$

where P_x and P_y are the x_W and y_W coordinates of the hallway point P in the world frame.

6.3.3 Computing Robot Orientation/Location from Multiple Correspondences

In general, PSEIKI will output many more image-edge to hallway-line correspondences than the minimum number needed by any of the approaches described in the preceding subsection for the calculation of orientation and location of the robot. Since it seems reasonable to assume that any estimates of robot orientation and location that use all the available data will be more robust compared to the estimates derived from just a couple of image edges, we need a method for combining the orientation/location results derived from the different possible subsets of the PSEIKI's output. We use the following six-step approach for this purpose.

Step 1: Note that in the output produced by PSEIKI, we have a belief value associated with each image-edge to hallway-line correspondence. Let's denote the belief value associated with the i^{th} image-edge by b_i. We now weight these belief values by the length of the edges, since we want the system to give greater credence to the orientation/location results that are produced by longer edges. We therefore recompute the belief values by combining the PSEIKI-supplied belief values with the lengths of the edges via the following formula:

$$b'_i = \begin{cases} \dfrac{b_i\, l_i}{L_i} & l_i < L_i \\ b_i & otherwise \end{cases} \tag{6.21}$$

where b''s are the new belief values, l's the lengths of the image edges, and L a threshold length.

Step 2: The image-edge to hallway-line correspondences whose associated b' values are below a threshold are then discarded.

Step 3: The image-edge to hallway-line correspondences are then classified according to the direction of the hallway line involved.

Step 4: For each of the image-edge to hallway-line correspondences involving horizontal hallway lines, the orientation θ_i of the robot is computed by the method described in Section 6.3.1.

Step 5: We now select those image-edge to hallway-line correspondences that involve hallway lines parallel to either the x_W-axis or the y_W-axis of the world coordinate frame. By using the "SINGLE LINE CORRESPONDENCE METHOD" described in Section 6.3.2 on those PSEIKI-supplied correspondences which involve hallway lines parallel to the y_W-axis, we obtain the x_W coordinate of the the robot base center in the world frame, in other words the x_W component of $\vec{D}_{tran}$ shown in Fig. 6.5. Similarly, by applying the same method to those correspondences that involve hallway lines parallel to the x_W-axis, we obtain the y_W component of the robot base center.

Step 6: Finally, the method described under "TWO LINE CORRESPONDENCES METHOD" is used to again compute the x_W and y_W coordinates of the origin of the robot coordinate frame using each pair of non-parallel hallway lines that satisfy the conditions of this method.

The final value of the orientation θ of the robot coordinate frame with respect to the world frame is now computed by taking a weighted average of the orientations calculated by the method of Step 4 and the orientation as supplied by the odometry on the robot. With θ_e denoting the odometry-supplied value for the orientation, the formula we use for computing the final value is

$$\theta_{final} = \frac{\sum_i \theta_i\, w_i + \theta_e}{\sum_i w_i + 1} \tag{6.22}$$

where the index i refers to the image edge which yielded the orientation estimate θ_i. The factors w_i incorporate the altered beliefs b'_i, as calculated by Eq. (6.21), and, at the same

time, give greater weight to those edges which are closer to the robot:

$$w_i = b'_i (n_x + n_y) \tag{6.23}$$

where n_x and n_y are the same quantities as in Eq. (6.2) -- they are the x_R and y_R components of the unit normal to the plane of sight formed by the image edge i in the robot coordinate frame. The reason for this additional weighting is as follows: Note that the magnitude of the slope of the line passing through the origin in Fig. 6.4 is n_x/n_y. As n_x and n_y get progressively smaller, any errors in their values will have a larger effect on their quotient, and therefore a larger effect on the solutions produced by the intersection of the line with the circle shown in Fig. 6.4. This implies that we must give greater weight to the orientation solutions that are produced by those image edges whose corresponding hallway lines on the floor are close to the robot, since for such edges n_z will be small and, relatively speaking, either or both of n_x and n_y large.

The final values of the x_W and the y_W coordinates of the robot base center are obtained by taking a weighted average of the results calculated in Steps 5 and 6 above and the odometry-supplied values for these coordinates. In the following formulas, we have used $D_{tran,x,e}$ and $D_{tran,y,e}$ for coordinates values as supplied by the odometry.

$$D_{tran,x,final} = \frac{\sum_i D_{tran,x,i} w_i + \sum_{jk} D_{tran,x,jk} w_{jk} + D_{tran,x,e}}{\sum_i w_i + \sum_{jk} w_{jk} + 1} \tag{6.24}$$

$$D_{tran,y,final} = \frac{\sum_i D_{tran,y,i} w_i + \sum_{jk} D_{tran,jk} w_{jk} + D_{tran,y,e}}{\sum_i w_i + \sum_{jk} w_{jk} + 1}$$

with w_i's given by Eq.(6.23), and w_{jk}'s set to

$$w_{jk} = b'_j b'_k \det_{j,k} \tag{6.25}$$

where b'_j and b'_k stand for the belief values associated with the correspondences involving the image edges j and k, respectively, and $\det_{j,k}$ is the value of the determinant in Eq. (6.18). The index i in the first term in the numerators and the denominators refers to each of the image edges used in Step 5 for the computation of one or the other coordinate values of the robot base center. The indices jk in the second terms in the numerators and the denominators refer to the pair of image edges used in Step 6 for an estimate. We weight the estimates produced in Step 5 by the factors w_i's, since as the plane of sight associated with an image edge becomes more and more horizontal, the calculation of d in Eq. (6.6) becomes more sensitive to errors in the angle ϕ; the factors w_i's give more weight to those estimates produced by Step 5 whose planes of sight bear a larger angle with the horizontal. Note that w_{jk} factors include the value of the determinant since the

smaller the value of $\det_{j,k}$, the less trust we can place in the corresponding solution.

6.4 EXPERIMENTAL RESULTS

We will now show some experimental results obtained from the camera image of Fig. 6.9. The schematic of Fig. 6.10 should help the reader identify the edges in the image of Fig. 6.9 that were used for the estimation of the orientation and the translation of the robot. Table 6.1 presents the results obtained for each edge, or each pair of edges, marked in Fig. 6.10. The first row in the table shows the odometry-supplied information; when the robot is at the position and orientation from which the image of Fig. 6.9 was taken, then, according to the odometer, the robot was at the location ($0.61m$, $3.96m$) with an orientation of 22.5^o. As the last column of the first row shows, the weight given to this information is one in the averaging formulas of Section 6.3.3.

The hallway line corresponding to the image edge A is parallel to the y_W axis of the world frame, so it should give an estimate for the x_W coordinate of the robot base center by the "SINGLE LINE CORRESPONDENCE METHOD" of Section 6.3.2. This estimate comes out to 0.71 m, which 10 cm greater than the odometry supplied information. The weight associated with this estimate, calculated by using Eq. (6.23), is 0.53. Note that since the hallway line corresponding to the image edge A is horizontal, this image edge can also be used for orientation calculation, which in this case yields 22.78^o.

Along similar lines, using the pair of image edges B and D, the method "TWO LINE CORRESPONDENCE METHOD" of Section 6.3.2 yields the estimate ($0.50m$, $4.11m$) for the position of the robot, the weight to be associated with this estimate being 0.15; and

When all the estimates are averaged by using the formulas of Section 6.3.3, we get the final result shown in the last row of the table.

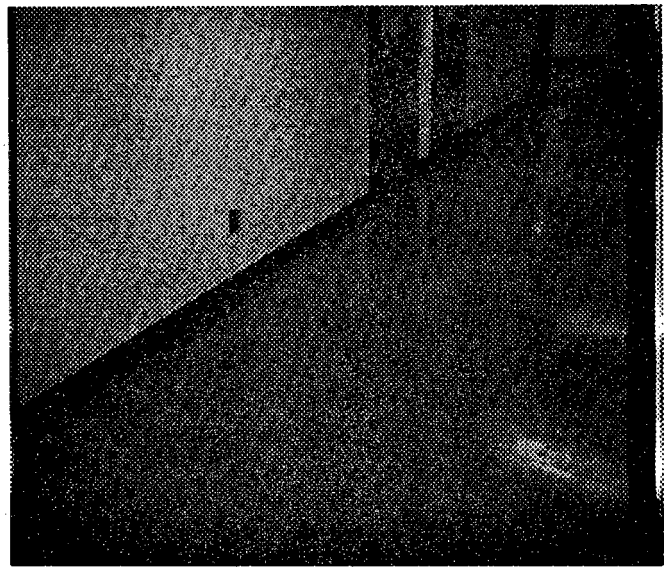


Figure 6.9: The image used for obtaining the self-location results of Table 6.1.

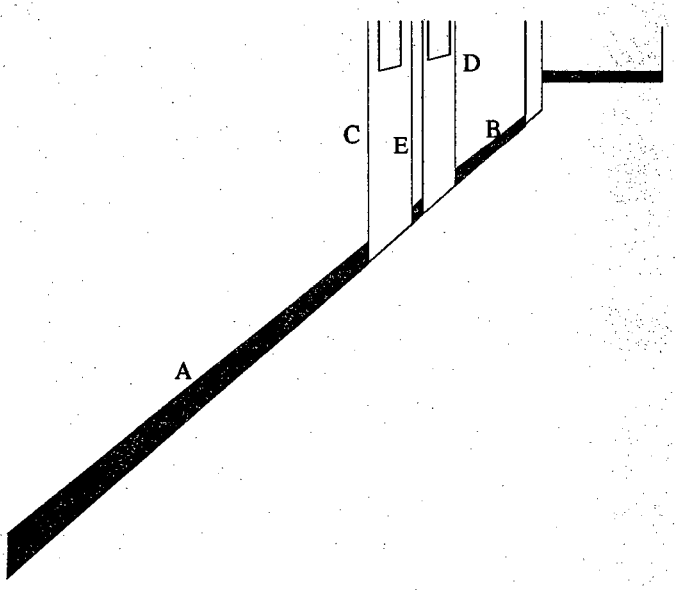


Figure 6.10: Segments used from the image of Fig. 6.9.

Table 6.1: Results of the self location program applied to the image of Fig. 6.9.

matches	X (m)	Y (m)	θ (deg)	weight
expected	0.61	3.96	22.5	1.0
A	0.71	-	22.78	0.53
B	0.69	-	22.33	0.54
A,C	0.55	3.83	-	0.23
A,D	0.55	3.96	-	0.16
A,E	0.55	3.88	-	0.19
B,C	0.50	3.94	-	0.23
B,D	0.50	4.11	-	0.15
B,E	0.50	4.01	-	0.19
Final	0.61	3.95	22.53	-

REFERENCES

[1] K.M. Andress and A.C. Kak, "Evidence Accumulation and Flow of Control in a Hierarchical Spatial Reasoning System", *AI Magazine*, Vol. 9, No. 2, pp. 75-95, 1988.

[2] K.M. Andress and A.C. Kak, "The PSEIKI Report -- Version 2", Purdue University, School of Electrical Engineering, TR-EE 88-9, 1988.

[3] A.C. Kak, B.A. Roberts, K.M. Andress and R.L. Cromwell, "Experiments in the Integration of World Knowledge with Sensory Information for Mobile Robots", *Proceedings IEEE International Conference on Robotics and Automation*, Vol. 2, pp. 734-741, 1987.

[4] C.L. Forgy, "OPS/83 User's Manual and Report", Production Systems Technologies, Inc., 1986.

[5] G. Shafer, "A Mathematical Theory of Evidence", Princeton University Press, 1976.

[6] A.C. Kak, K.M. Andress, and C. Lopez-Abadia, "Mobile Robot Self-location with the PSEIKI System", *AAAI Workshop on Robot Navigation,* AAAI Spring Symposium Series, Stanford, CA, 1989.

[7] A. C. Kak, K. M. Andress, C. Lopez-Abadia, M. S. Carroll, and J. R. Lewis, "Hierarchical Evidence Accumulation in the PSEIKI System and Experiments in Model-Driven Mobile Robot Navigation," *Proceedings of the Workshop on Uncertainty in Artificial Intelligence,* Windsor, Canada, 1989.

BIBLIOGRAPHY

BIBLIOGRAPHY

Manual of Photogrammetry, American Society of Photogrammetry, 1980.

K.M. Andress and A.C. Kak, "Evidence accumulation & Flow of Control in a Hierarchical Spatial Reasoning System", *AI Magazine*, pp. 75-94, Summer 1988.

K.M. Andress and A.C. Kak, "The PSEIKI Report -- Version 2", Purdue University, School of Electrical Engineering, TR-EE 88-9, 1988.

N. Ayache, O.D. Fagueras, F. Lustman and Z. Zhang, "Visual Navigation of a Mobile Robot: Recent Steps", *Proceedings IROS, Tokio*, pp. 651-658, 1988.

D. H. Ballard and C. M. Brown, *Computer Vision*, Prentice-Hall, 1982.

J.A. Bondy and U.S.R. Murty, *Graph Theory with Applications*, North-Holland, 1976.

R.A. Brooks, "A Robust Layered Control System for a Mobile Robot", *IEEE Journal of Robotics and Automation*, RA-2, pp. 14-23, April 1986.

R.A. Brooks, J.H. Connell and A. M. Flynn, "A Mobile Robot with Onboard Parallel Processor and a Large Workspace Arm", *Proceedings AAAI Conference*, pp. 1096-1100, 1986.

E.W. Dijkstra, "A note on two problems in connection with graphs", *Numer. Math.*, 1, pp. 269-271, 1959.

R. O. Duda and P. E. Hart, "Use of the Hough transformation to detect lines and curves in pictures", *Communications of the ACM*, vol. 15, pp. 1-15, 1972.

R. O. Duda and P. E. Hart, *Pattern Recognition and Scene Analysis*, Wiley, New York, 1973.

R. B. Eberlein, "An Iterative Gradient Edge Detection Algorithm", *Computer Graphics and Image Processing*, vol. 5, pp. 245-253, 1976.

A.E. Elfes, "A Sonar-Based Mapping and Navigation System", *IEEE Journal of Robotics and Automation*, RA-3(3), pp. 249-266, 1987.

A. M. Flynn and R.A. Brooks, "MIT Mobile Robots - What's Next?", *Proceedings IROS, TokioP, pp. 611-617, 1988.*

C.L. Forgy, "OPS/83 User's Manual and Report", Production Systems Technologies, Inc., 1986.

S. Ganapathy, "Decomposition of transformation matrices for robot vision", *Proc. Int. Conf. Robotics and Automation,* pp. 130-139, 1984.

Y. Goto and A. Stentz, "Mobile Robot Navigation: The CMU System", *IEEE Expert,* Winter 1987.

K. D. Gremban, C. E. Thorpe and T. Kanade, "Geometric Camera Calibration using systems of linear equations", *Proc. Int. Conf. Robotics and Automation,* pp. 562-567, 1988.

W. E. L. Grimson, *From Images to Surfaces,* M.I.T. Press, 1981.

P. V. C. Hough, "Method and means for recognizing complex patterns", U.S. Patent 3,069,654, 1962.

M. H. Hueckel, "An Operator that Locates Edges in Digital Pictures", *Journal of the ACM,* Vol. 18, pp. 113-125, 1971.

M. H. Hueckel, "A Local Visual Edge Operator Which Recognizes Lines", *Journal of the ACM,* Vol. 20, pp. 634-647, 1973.

J. Illingworth and J. Kittler, "The adaptive Hough transform", *Pattern Analysis and Machine Intelligence,* vol. PAMI-9, pp. 690-698, 1987.

A. Izaguirre, P. Pu, J. Summers, "A new development in camera calibration: Calibrating a pair of mobile cameras", *Proc. Int. Conf. Robotics and Automation,* pp. 74-79, 1985.

A. N. Jain and D. B. Krig, "A robust Hough transform technique", *Vision,* vol. 4, pp. 1-5, 1987.

A. C. Kak, "Depth perception for robots", *Handbook of Industrial Robotics,* pp. 272-319, John Wiley, New York, 1985.

A.C. Kak, B.A. Roberts, K.M. Andress and R.L. Cromwell, "Experiments in the Integration of World Knowledge with Sensory Information for Mobile Robots", *Proceedings IEEE International Conference on Robotics and Automation,* Vol. 2, pp. 734-741, 1987.

A.C. Kak, K.M. Andress and C. Lopez-Abadia, "Mobile Robot Self-location with the PSEIKI System", *AAAI Conference on Innovative Applications of Artificial Intelligence,* Stanford, CA, 1989.

R. K. Lenz and R. Y. Tsai, "Techniques for calibration of the scale factor and image center for high accuracy 3D machine vision metrology", *Pattern Analysis and Machine Intelligence,* vol. 10, pp .713-720, 1988

H. Li, M. A. Lavin and R. J. LeMaster, "Fast Hough Transform", *Proc. 3rd Workshop Computer Vision: Representation and Control,* pp. 75-83, 1985.

B. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision", *Proc. Image Understanding Workshop*, pp. 121-130, 1981.

D. Marr and T. Poggio, "A Computational Theory of Human Stereo Vision", *Proc. of Royal Society of London*, B, vol. 204, pp. 301-328, 1979.

D. Marr and E. Hildred, "Theory of Edge Detection", *Proceedings of the Royal Society of London*, B207, pp. 187-217, 1980.

H. A. Martins, J. R. Birk and R. B. Kelley, "Camera models based on data from two calibration planes", *Computer Graphics Image Processing*, vol. 17, pp. 173-180, 1981.

G. Medioni and R. Nevatia, "Matching images using linear features", *Pattern Analysis and Machine Intelligence*, vol. PAMI-6, no. 6, pp. 675-685, 1984.

G. Medioni and R. Nevatia, "Segment-Based Stereo Matching", *Computer Vision Graphics and Image Processing*, vol. 31, pp. 2-18, 1985.

L.S. McTamaney, "Real-Time Intelligent Control", *IEEE Expert*, Winter 1987.

H.P. Moravec, *Robot Rover Visual Navigation*, UMI Research Press, 1981.
H. P. Moravec, "The Stanford Cart and the CMU Rover", *Proceedings IEEE*, pp. 872-884, July 1983.

R. Nevatia and K. R. Babu, "Linear feature extraction and description", *Computer Graphics and Image Processing*, vol. 13, pp. 257-269, 1980.

R. Nevatia, *Machine perception,* Prentice-Hall, 1982.

N.J. Nilsson, "Shakey the Robot", Tech. Report 323, SRI AI Center, April 1984.

A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, Academic Press, 1980.

G. Shafer, "A Mathematical Theory of Evidence", Princeton University Press, 1976.

Y. Shirai, *Three-Dimensional Computer Vision*, Springer-Verlag, 1987.

R. Y. Tsai, "A versatile camera calibration technique for high accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses", *Proc. IEEE Conference on Computer Vision and Pattern Recognition,* pp. 363-374, 1986.

R. Y. Tsai, "A versatile camera calibration technique for high accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses", *IEEE Journal of Robotics and Automation,* vol. RA-3 NO. 4, pp. 323-344, 1987.

C.J. Van Wyk, *Data Structures and C Programs*, Addison-Wesley, 1988.

K. W. Wong, "Mathematical formulation and digital analysis in close range photogrammetry", *Photogrammetric Eng. Remote Sensing,* vol. 41, pp. 1355-1373, 1975.

Y. Yakimovsky and R. Cunningham, "A system for extracting three dimensional measurements from a stereo pair of TV cameras", *Computer Graphics Image Processing,* vol. 7, pp. 195-210, 1978.

APPENDIX

APPENDIX

DIJKSTRA'S MINIMUM-COST PATH FINDING ALGORITHM

Let G be a graph and let there be associated with each edge e of G a real number $w(e)$ called its weight. Then G together with these weights for the edges is called a *weighted graph*. If H is a subgraph of a weighted graph, the weight $W(H)$ of H is the sum of the weights $\sum_e w(e)$ over its edges. We will now present Dijkstra's algorithm that finds all the minimum-weight paths from any given node in a graph to all the other nodes [1].

Let $d(u, v)$ be the total weight associated with a minimum-weight path from a node u to a node v. Let's now consider the following situation: We are given a set $S \subset V$ of nodes, where V is the set of all nodes in G. Further, we are given a particular node $u_0 \in S$ and we are asked to find a minimum-weight path from u_0 to $\bar{S}$, where $\bar{S} = V - S$. Let $P = u_0 \cdots \bar{u}\, v$ be a minimum-weight path from u_0 to a node v in $\bar{S}$, where $\bar{u}$ is the node that occurs just before v on the path; clearly $\bar{u}$ is one of the neighbors of v. If P is a minimum-weight path from u_0 to node v in $\bar{S}$, then it must be the case that there exists a $\bar{u} \in S$, possible the same as u_0, such that $u_0 \ldots\ldots \bar{u}$ is shortest path from u_0 to $\bar{u}$; note that $\bar{u} \in S$ must be a neighbor of $v \in \bar{S}$. This fact is represented by the following equation:

$$d(u_0, \bar{v}) = d(u_0, \bar{u}) + w(\bar{u}\bar{v}) \tag{A.1}$$

We may now write the following expression for a minimum-weight path from the node u_0 to $\bar{S}$:

$$d(u_0, \bar{S}) = \min_{\substack{u \in S \\ v \in \bar{S}}} \{d(u_0, u) + w(uv)\} \tag{A.2}$$

That the weight of an optimum path from the given node u_0 to the set $\bar{S}$ of nodes can be expressed in the manner shown in Eq. (A.2) can serve as a basis for the following algorithm for computing minimum-weight paths. Starting with the set $S_0 = \{u_0\}$, we construct a sequence $S_0, S_1, \cdots, S_n$ of subsets of V such that the subset S_{i+1} is generated by opening all the nodes in S_i that are neighbors of the nodes in $\bar{S}_i$; the minimization of Eq. (A.2) then yields a node $v \in \bar{S}_i$ for a minimum path from u_0 to $\bar{S}_i$ and $S_{i+1} = S_i \bigcup \{v\}$. With this approach, at stage i of the computation, we have all the minimum-weight paths from u_0 to all the nodes in S_i. Computation is terminated when

S_n contains the destination node.

To elaborate, to construct the subset S_1, we find the node $u_1 \in \bar{S}_0$ that is "nearest" to u_0, in the sense that of all the edges meeting at node u_0 the weight associated with the edge u_0u_1 is a minimum. We set now $S_1 = \{u_0, u_1\}$. Now to construct S_2, we open both u_0 and u_1 and use Eq. (A.2) to look for a node from the set $\bar{S}_1$. The minimization in Eq. (A.2) will yield a node $v \in \bar{S}_1$ such that the weight $d(u_0,\bar{S})$ is minimum. S_2 will now be set to $\{u_0, u_1, v\}$. Note at this time, we know the minimum-weight paths from u_0 to all the remaining nodes in S_2. This computation continues until the entire graph is covered.

During the expansion of S_i into S_{i+1} by the discovery of the node $v \in \bar{S}_i$ via the minimization in Eq. (A.2), we deposit at the node v a back-pointer to v's neighbor $\bar{u} \in S_i$. These backpointers define a tree rooted at u_0 such that the path between u_0 and any other node in the tree constitutes the minimum-weight path from u_0 to that node.

The above algorithm suffers from the following computational inefficiency: When S_{i+1} is computed from S_i via the minimization in Eq. (A.2), we must determine the weights associated with the minimum-weight paths from u_0 to all those nodes in S_i whose neighbors are in $\bar{S}_i$. And, then when we compute S_{i+2} from S_{i+1}, we must do the same to all the 'boundary' nodes in S_{i+1}, a computationally wasteful procedure since many of the boundary nodes in S_{i+1} will be the same as in S_i. In Dijkstra's algorithm, this duplication in computation is eliminated by associating a number $l(u)$ with each node u in the graph; this number is computed in the following manner: Let's say we have already computed S_i and that we know the weight $d(u_0, u)$ associated with a minimum-weight path from the node u_0 to each node u in S_i. Prior to the computation of S_{i+1}, the value of $l(u)$ at each node u in S_i is equal to $d(u_0,u)$. Now, for the computation of S_{i+1}, we open each node of S_i that was not already opened during the construction of S_i -- these nodes will be characterized by the fact that some of their neighbors will be in $\bar{S}_i$. Let $v \in \bar{S}_i$ be a new node obtained by expanding a particular node u of S_i. Then we set $l(v)$ to

$$l(v) = \min_{u \in S_i}[d(u_0, u) + w(u, v)] \qquad v \in \bar{S}_i \tag{A.3}$$

where the minimization is with respect to all the possible neighbors, each denoted by u, in S_i of the node $v \in \bar{S}_i$. It is important to note that the value of $l(v)$ thus obtained is NOT necessarily the value of $d(u_0, v)$; the reason for this will be clear shortly. After we have computed $l(v)$ to all the node v in $\bar{S}_i$ that are the neighbors of the nodes in S_i, in keeping with Eq. (A.2), we then select that v that has associated with it the smallest $l(v)$ and add this node to S_i to form S_{i+1}; this smallest value of $l(v)$ then becomes $d(u_0,\bar{S}_i)$ in Eq. (A.2).

Eq. (A.3) tells us that the value $l(v)$ should be computed in an iterative manner, primarily because the node $v \in \bar{S}_i$ may have more than one neighbor in S_i. Any time we open one of v's neighbors in S_i, we compare the new possible value for $l(v)$ with the previously computed $l(v)$; the new value replaces the old if the former is smaller. This iterative approach to updating $l(v)$'s requires that initially the values of $l(u)$ for all the nodes of the graph be set to some large number, larger than any that might result from the calculations. Here is a step-by-step description of the algorithm:

Dijkstra's Algorithm:

1. Set $l(u_0) = 0$, $l(v) = \infty$ for $v \neq u_0$, $S_0 = \{u_0\}$ and $i = 0$.
2. For each $v \in \bar{S}_i$ that satisfies Eq. (A.2), replace $l(v)$ by $\min\{l(v), \overline{l(u)} + \overline{w(uv)}\}$, $u_i \in S_i$. If $l(v)$ is modified, mark u_i as the predecessor of v. Compute $\min_{v \in \bar{S}_i}\{l(v)\}$ and let v' be the vertex for which this minimum is attained. Set $S_{i+1} = S_i \cup \{v'\}$.
3. If $\bar{S}_{i+1} = \varnothing$, stop. Otherwise replace i by $i+1$ and go to step 2.
4. Find the shortest path between u_0 and v_0 by tracing back the predecessor of each vertex from v_0 to u_0.

Note that this algorithm assumes the weights of the edges to be positive. If negative weights are allowed, a negative weight cycle ψ might occur. If a path from u_0 to v_0 uses any part of ψ then we can construct a shorter path by following it but going once more around ψ; thus there are infinitely many paths from u_0 to v_0, none of which is the shortest. In [3] some algorithms are presented that can cope with negative weights.

REFERENCES

[1] E.W. Dijkstra, "A note on two problems in connection with graphs", *Numer. Math.*, 1, pp. 269-271, 1959.

[2] J.A. Bondy and U.S.R. Murty, *Graph Theory with Applications*, North-Holland, 1976.

[3] C.J. Van Wyk, *Data Structures and C Programs*, Addison-Wesley, 1988.