5-31-1991

Analysis and implementation of a navigation system using vanishing points in a generalized environment

Rolf Schuster
New Jersey Institute of Technology

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

Analysis and Implementation of a Navigation System with Vanishing Points in a Generalized Environment

by

Rolf Schuster

The development of accurate sensors is of crucial importance in navigation of mobile autonomous robots. The following master's thesis analyzes the use of vanishing points for robot navigation. Parallel lines in the environment of the robot are used to compute vanishing points which serve as a reference for guiding a robot. To accomplish the navigation tasks, three subtasks are to be performed: detection of straight lines, computation of vanishing points, and robot navigation with vanishing points.

An edge detection algorithm is presented that combines Sobel and Laplacian of Gaussian operators. The algorithm preserves the precision of the Laplacian of Gaussian operator while the Sobel operator is mainly used for filtering image noise. A method to determine the Laplacian of Gaussian kernel is described. Recursive subdivision is used to detect raw lines in the edges. Raw lines are approximated by straight lines using a least squares fit.

Several methods for detecting vanishing points are presented. The cross-product method as introduced by Magee and Aggarval is described in detail. The method is modified in order to make the detection of vanishing points appropriate for an indoor environment. The navigation section derives the properties of vanishing points under camera rotation and translation. Using these properties, the location of the vanishing points can serve as a reference for robot navigation. A model of the robot environment is defined, summarizing the minimal number of constraints necessary for the method to work.

Finally, the limitations as well as the advantages of using vanishing points in robot navigation are described.

Analysis and Implementation of a Navigation System using Vanishing Points in a Generalized Environment

by

Rolf Schuster

A Thesis

Submitted to the Faculty of the Graduate Division of the

New Jersey Institute of Technology

in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Electrical Engineering

Department of Electrical Engineering

May 1991

APPROVAL PAGE

Analysis and Implementation of a Navigation System with Vanishing Points in a Generalized Environment

by

Rolf Schuster

Dr. N. Ansari, Thesis Adviser

Assistant Professor, New Jersey Institute of Technology

Dr. E. Hou, Committee Member

Assistant Professor, New Jersey Institute of Technology

Dr. A. Banihashemi, Committee Member

Senior Researcher, Siemens Corporate Research, Princeton, New Jersey

BIOGRAPHICAL SKETCH

Author : Rolf Schuster

Degree : Master of Science in Electrical Engineering

Date : May, 1991

Date of Birth :

Place of Birth :

Education :

- BACHELOR OF SCIENCE in Telecommunications and Electronics, Fachhochschule für Technik, Mannheim, Germany, 1989
- MASTER OF SCIENCE in Electrical Engineering, New Jersey Institute of Technology, Newark, New Jersey, 1991

Experience :

- RESEARCH ASSISTANT, Siemens Corporate Research, Princeton, USA, Jul. 90 - May 91
- ELECTRICAL ENGINEER, Siemens AG, Bensheim, West Germany, Mar. 89 - Aug. 89
- TEACHING ASSISTANT, Fachhochschule für Technik, Mannheim, West Germany, Mar. 88 - Dec. 88
- ASSOCIATE ENGINEER, University of Swansea, Swansea, Great Britain, Mar. 86 - Aug. 86

to Heike -

hoping it was worth the wait ...

ACKNOWLEDGMENT

At this point I would like to express my thanks to Professor Nirwan Ansari for his strong personal and academic support and the considerable amount of time he has spent on our regular meetings. Special thanks goes to Professor Edwin Hou and Dr. Ali Banihashemi for serving as members of the committee.

Most of the research work was done at Siemens Corporate Research, a R&D company with excellent facilities in Princeton, New Jersey. I appreciate the opportunity of working there not only because of the outstanding research environment, but also because of the many very helpful specialists like Ali Banihashemi, Bruce Ladendorf and Keith Andress who often have contributed valuable ideas to my work. I am particular grateful to Ali Banihashemi for his guidance and time intensi · help especially with difficult parts of the research. Gus Tsikos, Don Guise, Ruth Weitzenfeld and Angelica Formento also assisted me with my work at Siemens Corporate Research.

Special thanks to Sukhmani Kaur who helped me with typing and formating the thesis. Finally I would like to thank all my friends for their moral support and useful hints as well as their patient accepting me in that situation.

Contents

List of Figures

Chapter 1

INTRODUCTION

It has always been the goal of science and engineering to understand nature and to use that understanding for the benefits of the human society. Today's technology provides many conveniences for our daily life. Many innovations and developments extend the capabilities of human beings beyond their natural limits. For example, technology enables us to travel fast from one place to the other and it enables us to communicate with each other across long distances.

Part of this general development in technology are attempts to build robotic systems that have human-like intelligence and are capable of moving independently in their environment. These robotic systems can, for examples, be used to perform hazardous tasks and monotonous work.

The cardinal problem of all robotic systems is how they access information about their environment. It is essential for the robot to be able to access information about its position, orientation or speed with respect to its environment. This can be achieved by using sensors to measure distance, angular turn, radiation and temperature. However, the current development in sensors and sensor integration is still far from reaching the capabilities of the human sensing system.

This is the motivation to analyze the usefulness of vanishing points in robot

navigation. The objective is to navigate a mobile robot based on the vanishing points of parallel lines in its environment. The overall analysis can be summarized in three steps. First, the robot detects the lines in its environment using a vision system together with an edge detection and line fitting algorithm. Second, the vanishing points are computed using the data of the detected lines . In the third step, the position of the vanishing points are integrated in the navigation process of the robot. This master's thesis presents all three steps with the theoretical analysis as well as the results of the implementation. Furthermore, a model of the robot environment is defined summarizing the essential assumptions which are necessary for the method to work.

Chapter 2

EDGE DETECTION

2.1 INTRODUCTION

Many theories and algorithms applied to high-level vision tasks assume that the pictures are already segmented, i.e., the desired features (lines, edges etc.) of the picture are enhanced. Hence edge detection is of crucial importance in Image Processing and Computer Vision. Various edge detectors can be classified into two broad classes, depending on their principle:

1. First derivative operators (Gradient operators) respond to an edge with a broad peak, which degrades the resolution of edge detection [HM86, p293].

2. Second derivative operators (Laplacian operators), which respond with a zero-crossing to an edge, are generally regarded as very precise (in a noise free environment). Since they use the second derivative, they enhance high frequency image noise.

Both types of operators have been studied extensively and their advantages and drawbacks are well known (see [GW87] for a good overview). The basic idea presented in this work is to combine the advantages of both types of operators.

Both the first and second derivative operators are applied separately to the original image. Then the results are combined by accepting only those zero-crossings in the result of the second derivative operator as an edge point, if at the same location, the result of the first derivative operator is above a certain threshold. Hence, the first derivative operator acts like a mask filter on the result of the second derivative operator. It eliminates all those zero-crossings in the result of the second derivative operator that are not located on a gradient with sufficient slope. Thus one can make use of the precision of the second derivative operator (zero-crossings) and "mask out" the high frequency image noise with the first derivative operator.

2.2 FIRST DERIVATIVE OPERATORS

The slope m of the one dimensional analog function $g(x)$ is defined as $m = dg(x)/dx$. In similar fashion we can approximate the vertical and horizontal gradient of a two-dimensional function $f(x, y)$ as

$$\begin{aligned} (\Delta_x f)(x, y) &= f(x-1, y) - f(x+1, y), \\ (\Delta_y f)(x, y) &= f(x, y+1) - f(x, y-1). \end{aligned}$$

These first order differences of $f(x, y)$ are commonly expressed as convolution operators which convolve f(x,y) with the patterns

$$\begin{bmatrix} 1 & 0 & -1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}.$$

Since these first order difference operators are odd order derivatives, they are not isotropic, i.e., rotation invariant (in the sense that rotating image h and then applying the operator gives the same result as applying the operator to image h and then

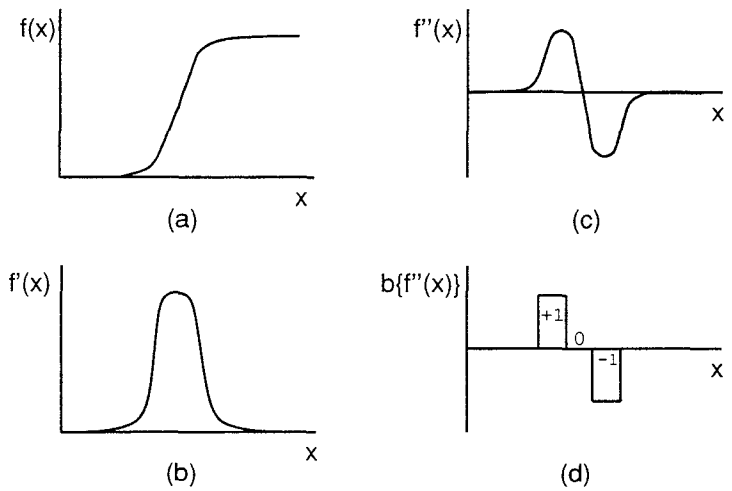

Figure 2.1: *Convolution kernel from the second derivative: (a) Blurred edge, (b)-(c) first and second derivative of (a), (d) approximation of second derivative in a convolution kernel.*

rotating the output). See [RK82a, page 238] for a precise definition of 'isotropic'. However, these first order difference operators can be made isotropic by taking the sum of the squares of the vertical component $\Delta_x f(x,y)$ and the horizontal component $\Delta_y f(x,y)$.

At this point it might be interesting to observe that we can derive the same convolution kernels using properties of pattern matching: if we define a template g according to the gray values of the edge to be detected, a match of g in an image h gives a possible location of an edge element. If we are interested in a sharp match response rather than in the actual gray level result of the match, it is advantageous to define template g as the second derivative of the edge [RK82b, page 43]. If we look at the two step edge with a cross section of $\ldots aaa[(a+b)/2]bbb\ldots$, the second order differences would turn out to be $\ldots\ 00[(b-a)/2]0[(a-b)/2]00\ \ldots$, which have values proportional to the template [1 0 -1] (see Figure 2.1). This template is the exact equivalence of the convolution kernels derived for the first order difference operator. The only difference between the two methods is that while edge detection

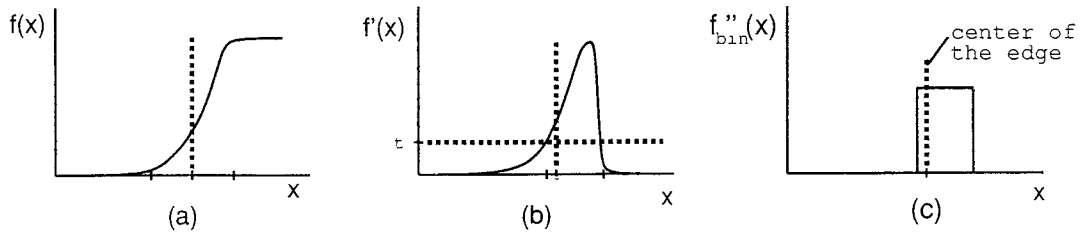

Figure 2.2: *Sobel convolution applied to an asymmetrical edge: (a) original edge, (b) result of Sobel convolution with original edge, (c) thresholded Sobel result.*

convolves the convolution kernels with the image, pattern matching correlates the template with the image (i.e., correlation does not involve flipping either the image or the template).

One can improve the convolution kernel by smoothing in the direction of the expected edge and assigning different weight-factors according to the distance to the center of the kernel. The resulting operator is called the Sobel operator. The vertical and horizontal convolution masks are defined by

$$\frac{1}{4}\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad \text{and} \quad \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}.$$

The main disadvantage of the first derivative operators is that it locates the edge with a relatively broad peak. This makes thresholding and perhaps even thinning necessary, and limits the resolution of detecting the edge. Figure 2.2 illustrates these properties as well as the fact that in the case of asymmetrical edges, the center of the original edge is not necessarily at the center of the detected broad peak.

2.3 SECOND DERIVATIVE OPERATORS

The second derivative operator can be derived in similar fashion: the second derivative of the function $g(x)$ is given by $d^2g(x)/dx^2$. Hence the second derivative of the two dimensional discrete function $f(x,y)$ can be approximated by the difference between the first order differences on the right and left side of the center point:

$$\begin{aligned}(\Delta_x^2 f)(x,y) &= [f(x+1,y) - f(x,y)] - [f(x,y) - f(x-1,y)],\\ (\Delta_y^2 f)(x,y) &= [f(x,y+1) - f(x,y)] - [f(x,y) - f(x,y-1)].\end{aligned}$$

Since these are even order derivatives one can simply add the vertical and horizontal components to form an isotropic operator [RK82a, page 238], i.e., rotation invariant operator (Laplacian operator):

$$\begin{aligned}(\bigtriangledown^2 f)(x,y) &= (\Delta_y^2 f)(x,y) + (\Delta_x^2 f)(x,y)\\ &= [f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1)] - 4f(x,y)\end{aligned}$$

which is the digital convolution of f(x,y) with the kernel

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

From this it is obvious that the Laplacian operator takes the difference between gray value at (x,y), which is the center point of the kernel, and the average in the neighborhood of (x,y) [RK82b, page 242]. This operator has the main disadvantage of responding sharply to single pixel noise. Like the first derivative operator, one can improve this operator by implementing smoothing capabilities. Here we use the second derivative of a Gaussian Distribution Function (Figure 2.3(b)). The mathematical equations for the continuous and one dimensional case are given by

$$G(x) = \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

and

$$G''(x) = \frac{1}{\sqrt{2\pi}\,\sigma}\left[\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2}\right] e^{-\frac{x^2}{2\sigma^2}}.$$

Expanding $G''(x)$ into a two dimensional function $G''(x,y)$ gives us the continuous Laplacian of Gaussian (LoG):

$$G''(x,y) = \frac{1}{2\pi\,\sigma^4}\left[\frac{x^2+y^2}{\sigma^2} - 2\right] e^{-\frac{x^2+y^2}{2\sigma^2}}.$$

Now the objective is to use $G''(x,y)$ to define the convolution kernel for the Laplacian Convolution.

2.3.1 Determining the Laplacian Convolution Kernel

The following procedure is used to determine a Laplacian kernel:

1. Choose kernel size and constant σ in $G''(x,y)$.
2. Quantize the continuous LoG $G''(x,y)$ into the discrete LoG $G''_d(i,j)$ with i, j being integer values [HL86].
3. Define kernel values such that the sum of all values is zero.

First we have to select the size of the Laplacian convolution kernel based on the desired edge detection as well as the hardware and the time constraints. The constant σ, which determines the shape and the extent of the LoG $G''(x,y)$, has to correspond to the size of the kernel in the sense that it has to be possible to approximate the shape of $G''(x,y)$ within the kernel. Huertas and Medioni [HM86] stated that it is sufficient if the kernel covers $\pm 3\sigma$ from the origin of the continuous Laplacian $G''(x,y)$. Since 99.73% of the area of a one dimensional Gaussian lies between $\pm 3\sigma$, the area of the second derivative must be very close to zero.

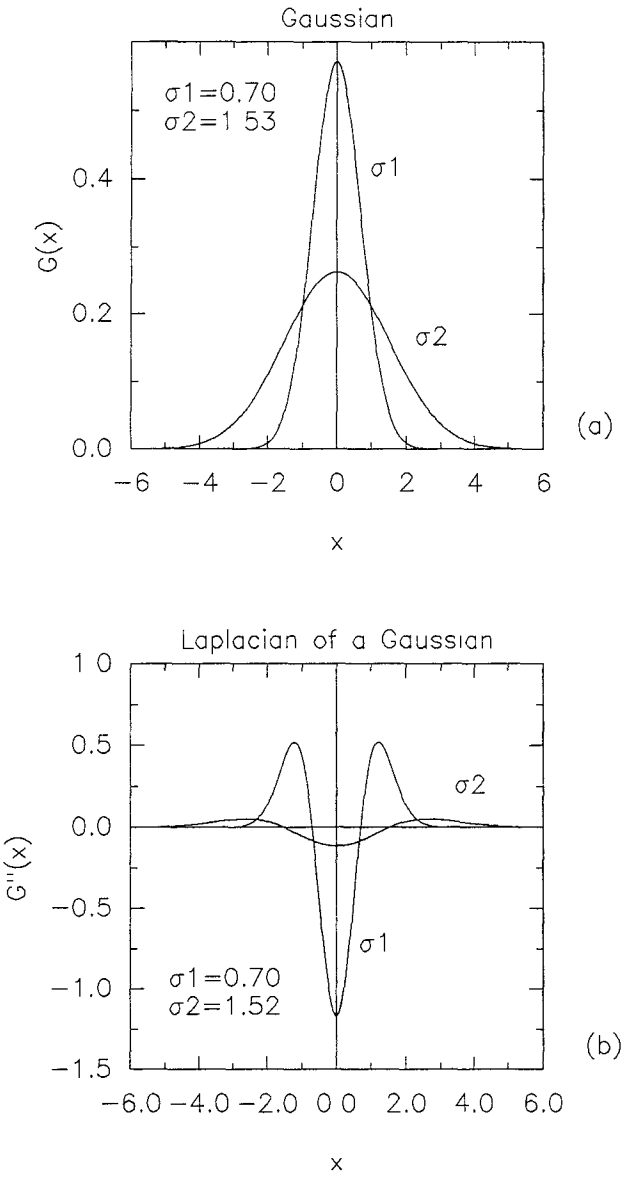

Figure 2.3: *Gaussian Functions: (a) Gaussian Distribution Function $G(x)$, (b) Second Derivative $G''(x)$.*

The second step is to quantize the continuous LoG $G''(x,y)$ into the discrete LoG $G''_d(\imath,\jmath)$ where i and j are integer values. The straight forward way to do this is to evaluate $G''(x,y)$ at the grid points $(\imath,\jmath)$ with

$$G''_d(\imath,\jmath) = G''(\imath,\jmath).$$

and round it to the nearest integer. This method is not very accurate and has the consequence that the resulting values of $G''_d(\imath,j)$ do not sum up to zero any more, and hence corrections become necessary. If higher accuracy is desired, block averages of the continuous LoG $G''(x,y)$ can be used to compute the discrete LoG $G''_d(\imath,\jmath)$ using

$$G''_d(i,j) = \int_{\imath-1/2}^{\imath+1/2} \int_{\jmath-1/2}^{\jmath+1/2} G''(x,y)dxdy \ .$$

See [HL86] and [Bre84] for details on quantization with bilinear interpolation and higher order interpolants.

In the third step, we have to ensure that the sum of all the elements of the LoG kernel is equal to zero, to avoid any bias when performing the LoG convolution. If the kernel elements do not sum up to zero after the quantization, we have to adjust their values. To further illustrate these findings we perform Steps 1 to 3 in an example:

1. Hardware and time constraints are assumed to limit the kernel size to 7. For this kernel size, a good approximation of the shape of the LoG can be achieved with $\sigma = 1$.

2. Quantize the continuous LoG with point evaluations $G''_d(\imath,\jmath) = G''(\imath,j)$. Scale the resulting values by 60, and then round them off to the nearest integers.

3. Alter some kernel values to make the sum of the kernel values equal to zero.

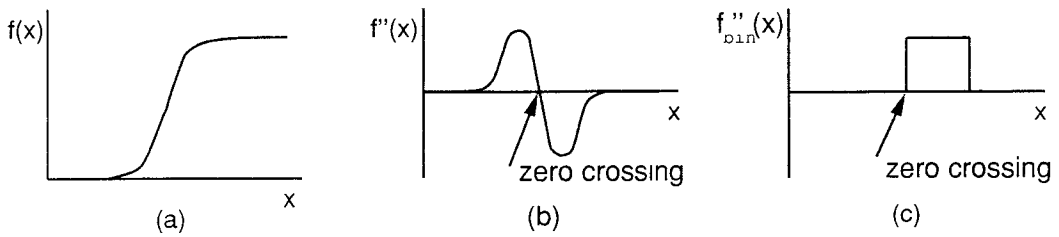

Figure 2.4: *Laplacian operator applied to an edge: (a) original edge, (b) result of LoG convolution, (c) binary representation of (b) with thresholding at 0.*

The Laplacian convolution kernel corresponding to the above example is:

$$\begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 & 2 & 1 & 0 \\ 1 & 2 & 0 & -6 & 0 & 2 & 1 \\ 1 & 3 & -6 & -20 & -6 & 3 & 1 \\ 1 & 2 & 0 & -6 & 0 & 2 & 1 \\ 0 & 1 & 2 & 3 & 2 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

2.3.2 Properties of the Laplacian Operator

Analyzing the Laplacian of Gaussian kernel (LoG) we find that it performs weighted averaging in the center of the kernel (negative kernel values) as well as in the neighborhood of the center (positive kernel values). Convolving an image $f(x, y)$ with that kernel results in the smoothed second derivative $f''(x, y)$ of that image.

Note that the values of $f''(x, y)$ change their sign (zero crossings) whenever a curvature change in the original $f(x, y)$ occurs (Figure 2.4). This means that we get positive values on one side of the edge and negative values on the other side of the edge and zero in between (only in case of very low contrast edges). In order to detect the zero-crossings, the LoG result should be thresholded at 0 (with $f''_{bin}(x, y) = 1$ if $f''(x, y) < 0$, and $f''_{bin}(x, y) = 0$ otherwise). Hence the zero-crossings provide a precise and blur-free means of detecting an edge.

2.4 COMBINING FIRST AND SECOND DERIVATIVE OPERATORS

From the analysis of the first and second order operators it is apparent that both operators have significant drawbacks. Although relatively stable with respect to image noise, the Sobel operator only provides an approximate location of the edge. The Laplacian on the other hand detects the edge location precisely (zero-crossings), but it is very susceptible to image noise. Zero crossings caused by image noise are thus falsely detected as edges. Hence we need additional constraints in order to extract more precise edges in noisy images using the second derivative operator.

Different types of constraints can be applied (see [HL86] for an overview). For example, we can "and" the zero-crossing contours of several successive scales of resolution, an approach which requires much more computational effort. Another possibility is to use the measure of the slope of the LoG image at the zero-crossing as a constraint. A zero-crossing is accepted as edge point, if the Sobel operator applied to the LoG image is above a threshold at the location of the zero-crossing.

Our approach uses the result of the Sobel operator applied to the original image as the constraint for the zero-crossings. We detect the zero-crossings in the LoG of the original image. A zero-crossing is accepted as the location of an edge point, if at the same location the result of the Sobel convolution with the original image is above a threshold. Figure 2.6 illustrates the algorithm in a block diagram.

The algorithm can be summarized in five steps:

1. Convolve original image $f(x, y)$ with vertical and horizontal Sobel kernels (see Chapter 2.2). The vertical and horizontal components are normalized and their absolute values are added. The result is called the Sobel image $f_S(x, y)$.

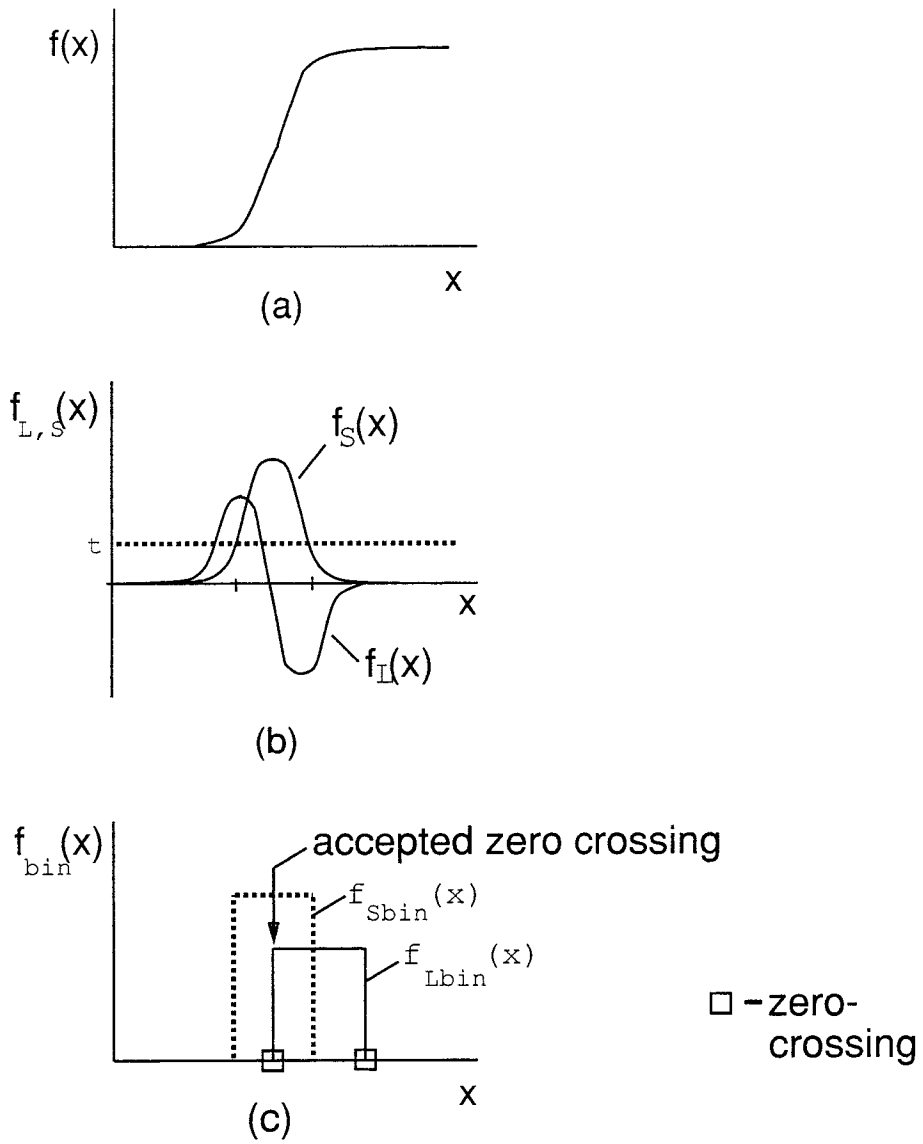

Figure 2.5: *Sobel and Laplacian operator applied to an edge: (a) original edge, (b) result of Sobel and LoG convolution, (c) accepted zero-crossing in binary representation of (b).*

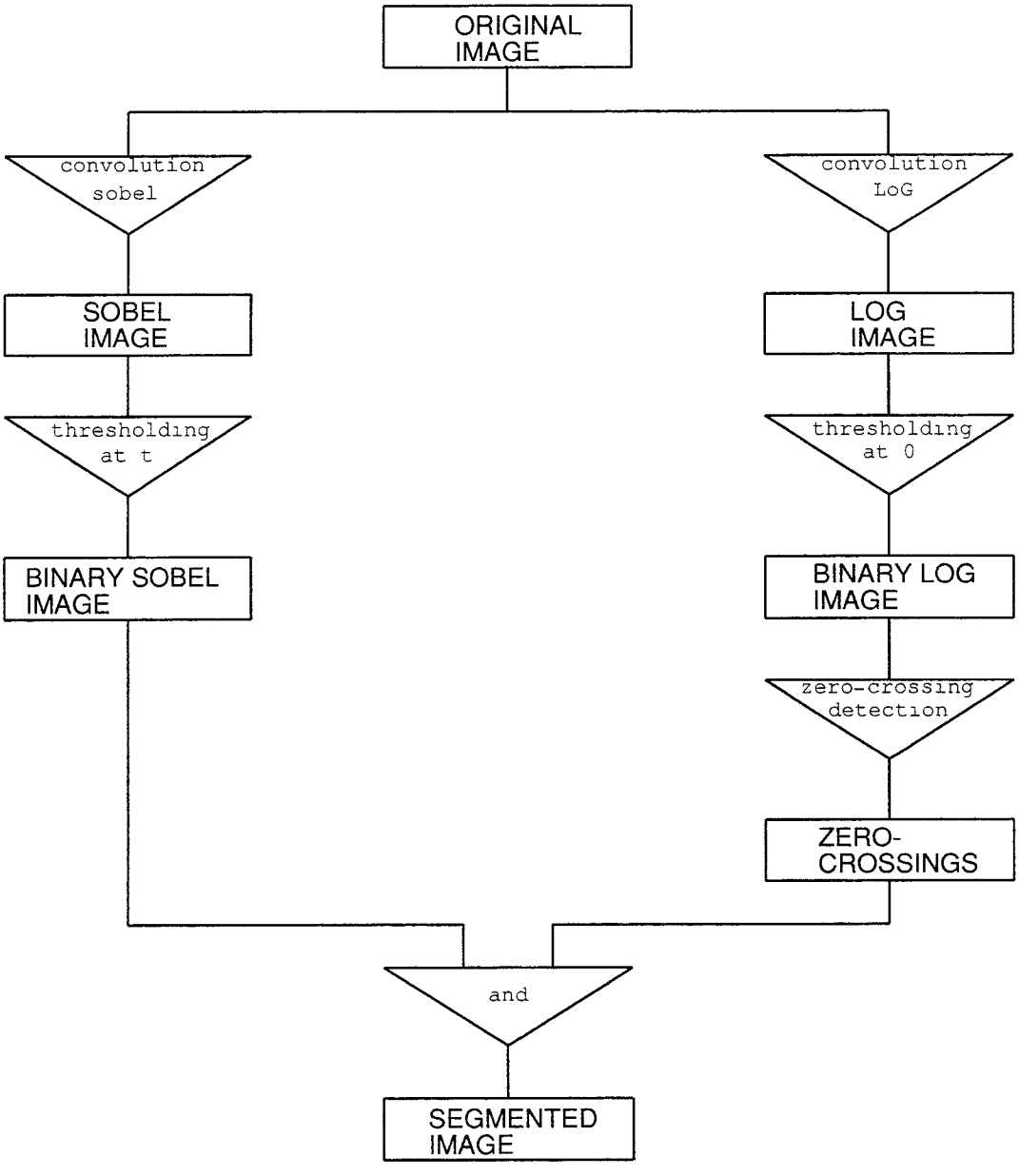

Figure 2.6: *Simplified block diagram of the edge detection algorithm.*

2. Convert $f_S(x,y)$ in a binary image $f_{Sbin}(x,y)$ (binary Sobel image) by applying a threshold t:

$$f_{Sbin}(x,y) = \begin{cases} 1 & \text{if } f_S(x,y) > t \\ 0 & \text{otherwise} \end{cases}$$

3. Convolve original image $f(x,y)$ with LoG kernel (see Chapter 2.3). We call the result LoG image $f_L(x,y)$.

4. Convert $f_L(x,y)$ in a binary image $f_{Lbin}(x,y)$ (binary LoG image) by thresholding at 0:

$$f_{Lbin}(x,y) = \begin{cases} 1 & \text{if } f_L(x,y) < 0 \\ 0 & \text{otherwise} \end{cases}$$

5. Scan every 8th row of the binary LoG image $f_{Lbin}(x,y)$ until a zero-crossing is found. If at that location the Sobel image is above a threshold ($f_{Sbin}(x_0,y_0) = 1$), this point is accepted as the start point of an curve (at (x_0,y_0)).

   Starting at (x_0,y_0) a tracing routine is used to search for consecutive locations which meet the two conditions: for each location, (1) a zero-crossing is found in the binary LoG image, and (2) the gray level in the Sobel image exceeds the threshold t. The x-, y-coordinates of the points which fulfill both requirements are stored in an array and labeled with the name of that curve. Assuming that (x_n,y_n) and (x_{n+1},y_{n+1}) are consecutive points on the curve, then the algorithm searches for the next point (x_{n+2},y_{n+2}) in the 8-neighborhood of (x_{n+1},y_{n+1}). This search scans the 8 neighboring points of (x_{n+1},y_{n+1}) (except (x_n,y_n), the previous point on the curve), until a point is found that meets both requirements. Hence there are a maximum of seven neighboring points scanned. The tracing algorithm reaches the end of a curve if none of the seven neighboring points meets both requirements.

   If the end of an curve is reached, the algorithm resumes the 8th row scan at the point following the start point (x_0,y_0+1). The search for the next start

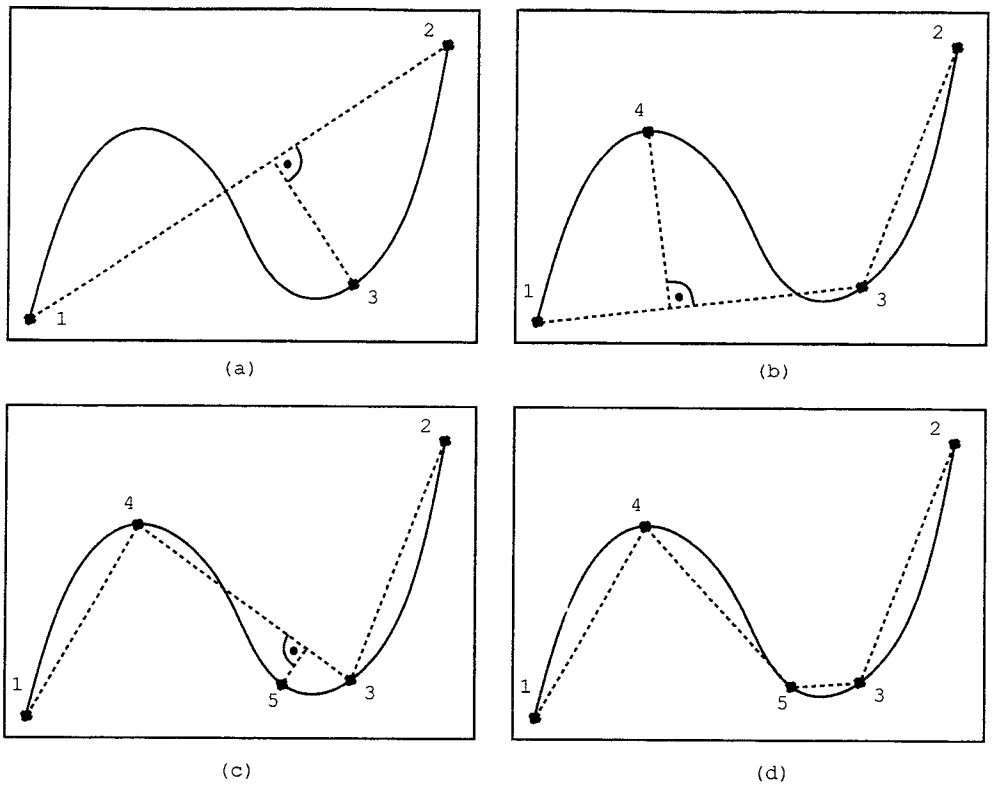

Figure 2.7: *Recursive Subdivision for line fitting.*

point of a curve begins. The end of the 8th row scan occurs when the bottom of the image is reached.

From the above discussion and the block diagram (Figure 2.5), it is clear that this algorithm still preserves the precision of the second derivative operator (LoG) but uses the first derivative operator (Sobel) as a constraint to mask out those zero-crossings resulting from image noise.

2.5 LINE FITTING

For many applications of image processing like object recognition or detection of vanishing points, it is important to find straight lines in an image. The proposed

technique for line detection assumes that the image is segmented, i.e., the curves are detected. The approach uses two steps to detect which parts of the curves can be considered straight lines. The first step does recursive subdivision of the curve and the second step performs a least-squared-error fitting.

The x, y-locations of the curve points are the input data for the recursive subdivision. Recursive subdivision is used to break up a curve segment into many smaller curve segments. A curve segment is split at the point which has the largest perpendicular distance to a straight line connecting the start and end point of the curve segment. This breaking up of curve segments is applied recursively until the average distance between the curve segment and the straight line connecting the start and end point of the curve segment is less than a threshold d. A curve segment which cannot be divided any further is called a raw line.

Figure 2.7 illustrates the recursive subdivision. Determine a straight line between point 1 and point 2. Find the point on the curve segment that has the largest perpendicular average distance to the line connecting the end points. Since the average distance is larger than the threshold d, the curve segment is split up at point 3. In Figure 2.7(b), there are now two straight lines connecting the start and end points of the two curve segments. No point on the curve segment between points 2 and 3 has a perpendicular average distance on to the straight link that exceeds the threshold d. Therefore, this curve segment is considered as a raw line. Following this procedure in the same manner, new break points (points 4 and 5) are detected as shown in Figures 2.7(c) and 2.7(d).

The second step of the line detection algorithm fits a straight line through the set of points on the raw line. This is done by using a least squares fitting

technique. The parameters m and b of the straight line

$$y = mx + b \tag{2.1}$$

can be obtained by minimizing the associated average squared error E. This error E is given by

$$E = \sum_{i=1}^{N} (mx_i + b - y_i)^2 \tag{2.2}$$

where (x_i, y_i) are the coordinates of points in the set, and N is the total number of points in the set. Refer to [SS88] and [Rei91] for further discussion. For many applications of line fitting very short line segments are not useful. Therefore, a line segments is accepted only if its length exceeds n pixels. Small segments are discarded. Figure 2.11 shows the results of applying the algorithm to a segmented image [1]. In this case, the average distance threshold d is defined to be 0.5 pixels. and the minimum line length n is set to be 40 pixels.

2.6 RESULTS AND CONCLUSION

The Figures 2.8-2.11 show the results of applying the edge detection and line fitting algorithm to a test image (Figure 2.8, 2.9 (top images)) which contains low and high contrast edges, straight and curved contours, and open and closed borders.

The images in Figure 2.8-2.10 demonstrate clearly how precisely the edge detection algorithm detects the various types of edges in the original image, and how effectively the noise can be eliminated. The threshold applied to the Sobel operator adjusts the sensitivity of the algorithm to the noise. A low threshold causes more "noisy" zero-crossings to be accepted as edge points. A high threshold filters out

[1] The program for the line fitting algorithm has been written by Dr. A. Banihashemi, Siemens Corporate Research

virtually all zero-crossings resulting from image noise along with some desirable zero-crossings. Hence there are contradicting requirements for the threshold. In that situation we can add another constraint for noise filtering: since image noise is usually local, a minimum length of a curve is required to be accepted as an edge.

The images in Figure 2.11 illustrate the result of the line fitting algorithm applied to the curves detected by the combined Sobel and LoG-operators.

The edge detection algorithms are implemented in the PC-based MATROX vision system, which provides the digitization as well as low level image processing operations. The images are taken by a SONY CCD camera with a zoom lens (12.5 - 75mm). The program for both the edge detection and the line fitting algorithm are is written in C.

As demonstrated in the resulting pictures (Figures 2.8-2.11), the algorithms appears to be appropriate for many applications when both high resolution edge detection and good noise suppression are required.

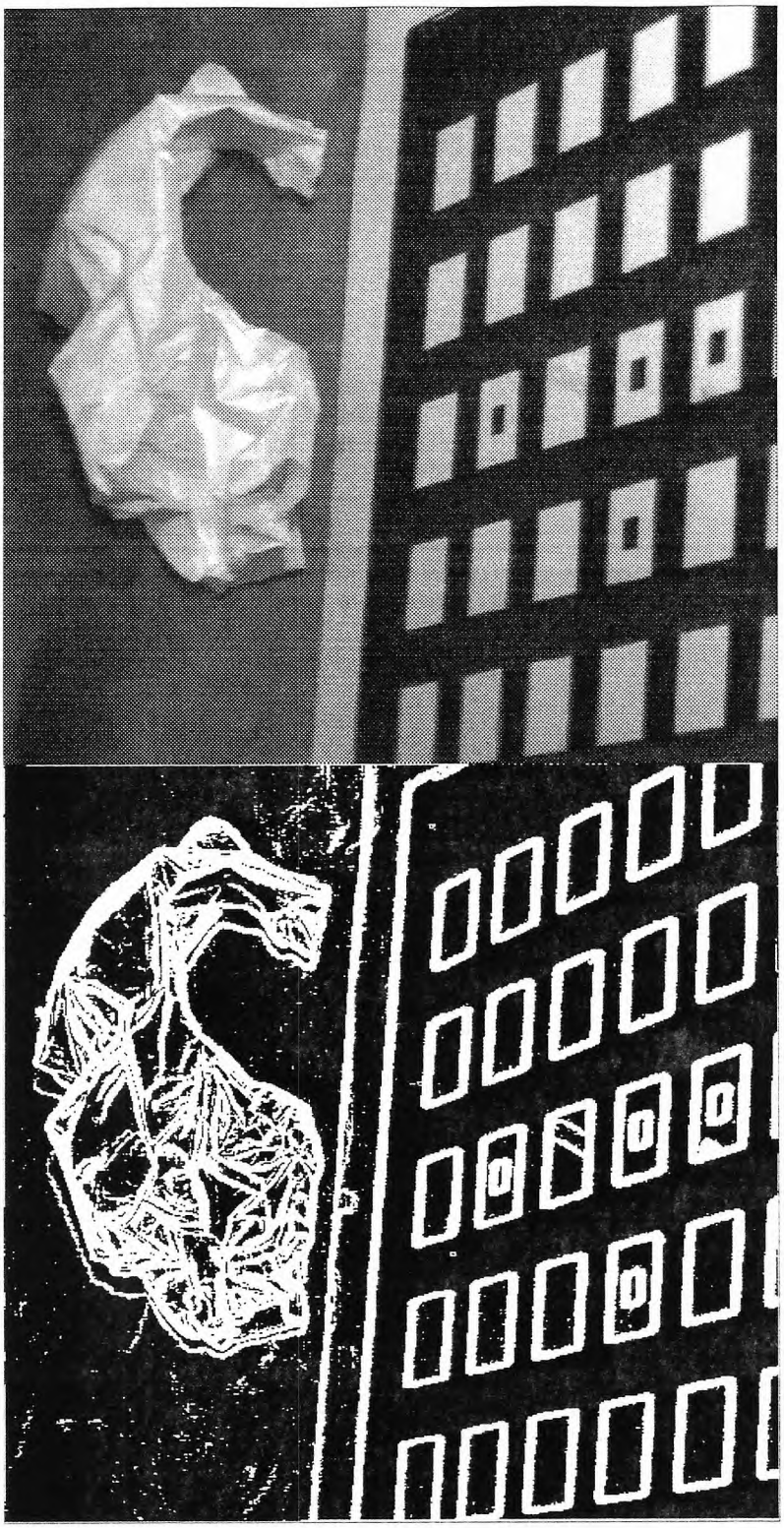

Figure 2.8: *The top image is the original image, and the bottom image results from applying the Sobel operator and using a threshold of* $t = 30$ *on the original image.*

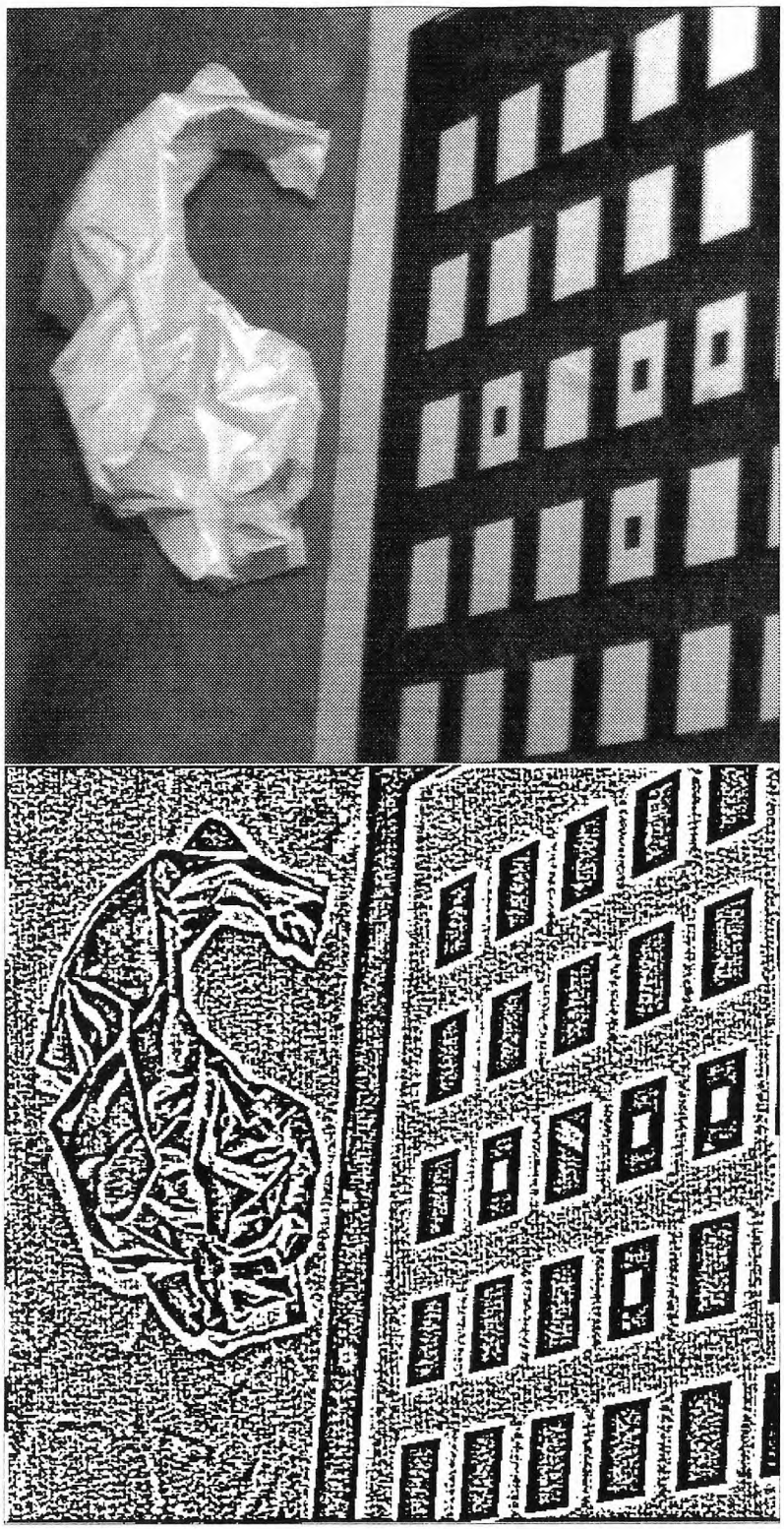

Figure 2.9: *The bottom image results from applying the LoG operator and using a threshold of* $t = 0$ *on the original image (top image).*

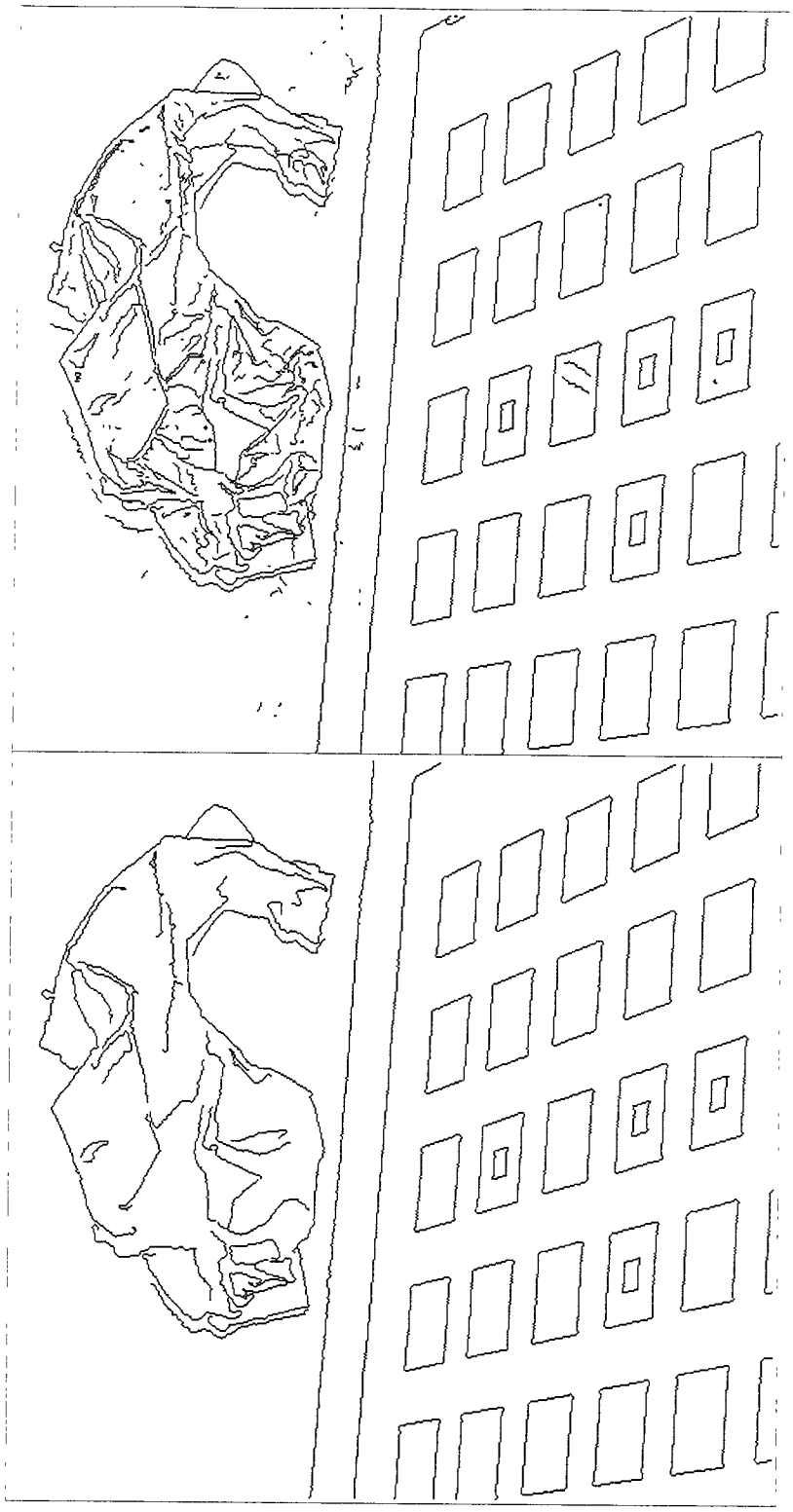

Figure 2.10: *Combining the results of both, the Sobel and the LoG operators yields the top image. Discarding short segments results in the bottom image.*

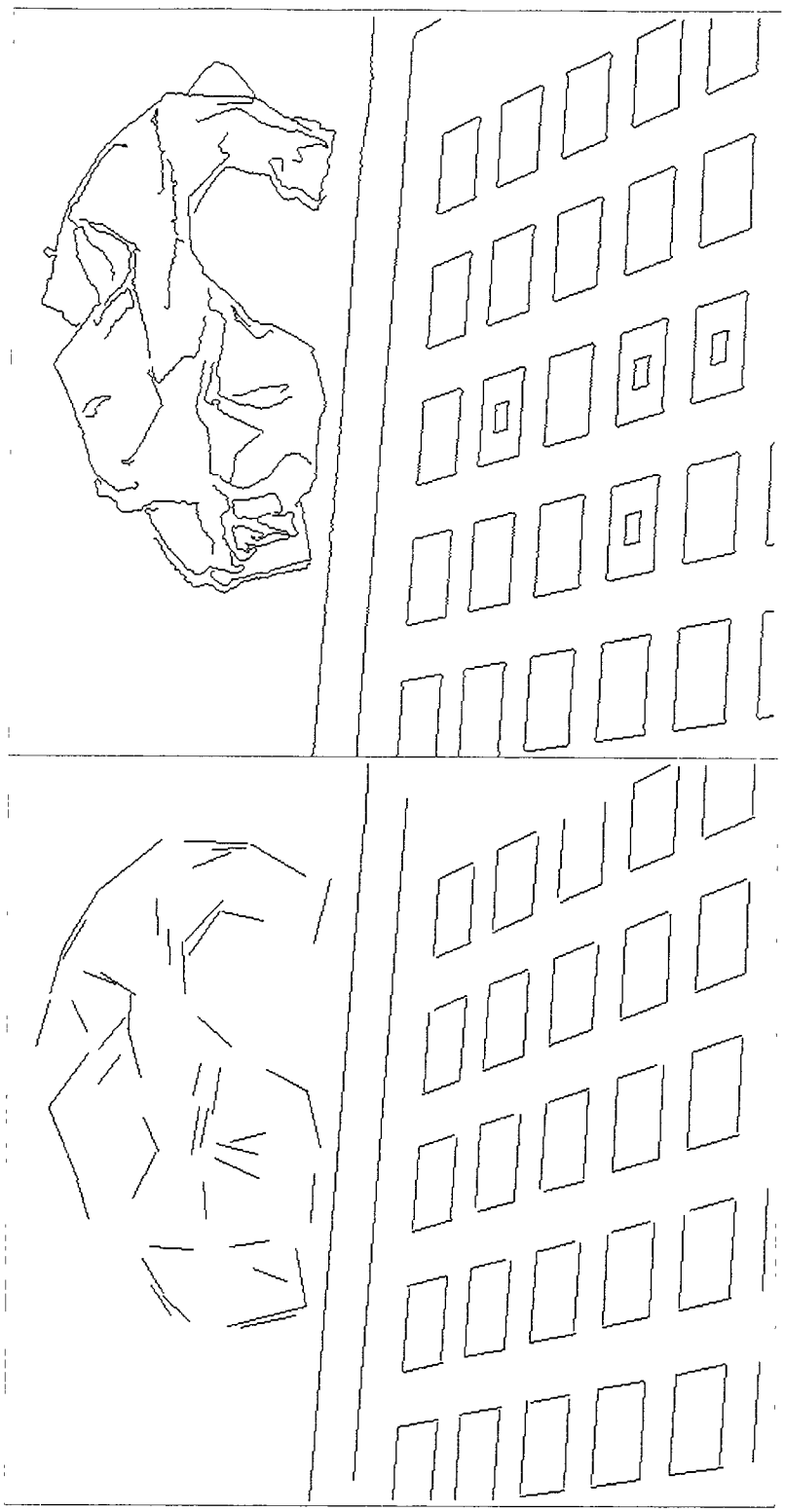

Figure 2.11: *Applying the line fitting algorithm to the detected curve segments in the top image results in the bottom image.*

Chapter 3

DETECTION OF VANISHING POINTS

3.1 INTRODUCTION

A fundamental problem in computer vision is how, given a two dimensional image, to derive information about the three-dimensional space. The problem is difficult because under perspective projection not only is the depth lost, but also the length and orientation of objects are also not invariant.

One method to derive information about three dimensional space from two dimensional images is by finding the vanishing points. Given two parallel lines on a plane in three-dimensional space, their projections onto the image plane intersect at a vanishing point, which provides information about the direction of the lines and provides a constraint on the orientation of the plane. Two such independent constraints determine the orientation of the plane uniquely. This method of deriving information about three-dimensional space from vanishing points combined with *a priori* knowledge of the three-dimensional space, is frequently used in camera calibration [CT90], three dimensional measurement [WR84] and robot navigation

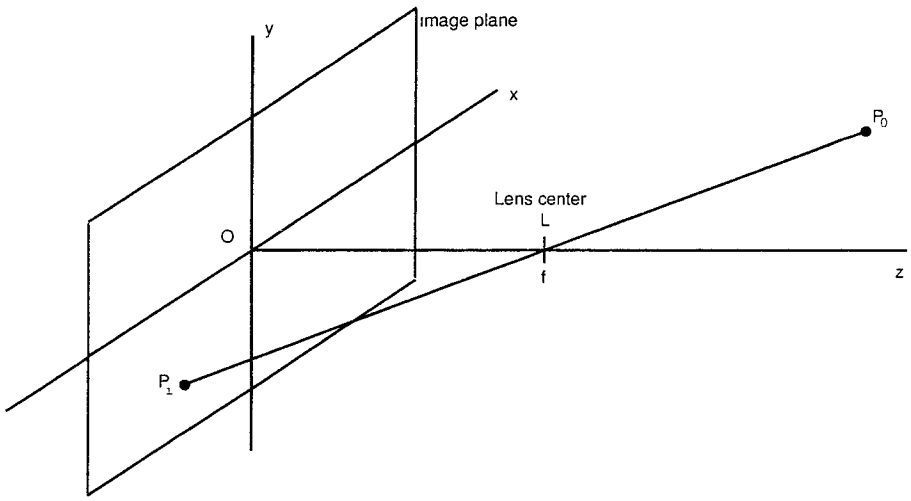

Figure 3.1: *Camera model and imaging transform of point* P_0.

[Bad74].

3.2 PROPERTIES OF IMAGING

The concept of vanishing points is based on the imaging transformation from real three-dimensional space into the two-dimensional image plane of the camera. In order to understand vanishing points, we first have to explore the properties of imaging.

3.2.1 The Imaging Transform

We define the camera coordinate system with the z-axis in the direction of the optical axis of the camera and the xy-plane in the image plane (Figure 3.1). In this coordinate system the center of projection is located at the origin O, and the optical center L of the lens is at the point $(0, 0, f)$, where f is the focal length of the lens. In this camera model, the imaging transform simply maps any point

P_0, in the three-dimensional space (scene point), through L into point P_i, in the two-dimensional image plane (image point). Then by similar triangles we have

$$x_i = -\frac{f\,X_0}{Z_0 - f} \quad , \quad y_i = -\frac{f\,Y_0}{Z_0 - f}\,, \tag{3.1}$$

where P_0 is defined as $(X_0,\ Y_0,\ Z_0)$ and P_i as $(X_i,\ Y_i,\ Z_i)$. The negative signs in the equations indicate that the image is inverted. It is important to note that this transform is non-linear since it involves a division by the variable Z. It is convenient to express transforms as matrices. For this transform we use homogeneous coordinates which express the Cartesian coordinates of the vector $\mathbf{w} = (X_0\,Y_0\,Z_0)^T$ to the scene point P_0 as $\mathbf{w_h} = (kX_0\,,\ kY_0\,,\ kZ_0\,,\ k)^T$, where k is an arbitrary, non-zero constant. Using homogeneous coordinates we can define the *imaging transformation matrix*

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{-1}{f} & 1 \end{bmatrix}. \tag{3.2}$$

Then the product $\mathbf{I}\,\mathbf{w_h}$ yields a vector that we denote by $\mathbf{c_h}$:

$$\mathbf{c_h} = \mathbf{I}\,\mathbf{w_h} \tag{3.3}$$

$$\mathbf{c_h} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{-1}{f} & 1 \end{bmatrix} \begin{bmatrix} kX_0 \\ kY_0 \\ kZ_0 \\ k \end{bmatrix} = \begin{bmatrix} kX_0 \\ kY_0 \\ kZ_0 \\ \frac{-k\,Z}{f} + k \end{bmatrix}. \tag{3.4}$$

The elements of $\mathbf{c_h}$ are the camera coordinates into homogeneous form. Converting $\mathbf{c_h}$ in Cartesian coordinates (dividing each of the first three components of $\mathbf{c_h}$ by the fourth) yields

$$\mathbf{c} = \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} = \begin{bmatrix} -\frac{f\,X_0}{Z_0 - f} \\ \frac{f\,Y_0}{Z_0 - f} \\ \frac{f\,Z_0}{Z_0 - f} \end{bmatrix}, \tag{3.5}$$

where vector **c** represents any point in the camera coordinate system in Cartesian form. The first two components of **c** are the $(x_i,\ y_i)$ coordinates in the image plane of the projected scene point $(X_0,\ Y_0,\ Z_0)$, as shown earlier by Equation 3.1. The third component of **c** is of no interest in terms of the transform from the three-dimensional space into the two dimensional image plane (it acts as a free variable in the inverse imaging transform [GW87]). The imaging transform maps many scene points P_n into one image point $(x_i,\ y_i)$. Namely, all the scene points P_n on the line through $(x_i,\ y_i,\ 0)$ and $(0,\ 0,\ f)$ correspond to the image point $(x_i,\ y_i)$. For further derivation it is useful to define the following property of the imaging transform:

PROPERTY 1: Straight lines in scene space, map into straight lines on the image plane [RK82b].

PROOF: Any straight line l in three-dimensional space and the point $(0,\ 0,\ f)$ (center of the lens) can be used to define a plane Λ. It is obvious that the intersection of the plane Λ with the image plane represents the projection of line l and this intersection is also a straight line, named l'. For the mathematical proof, consider line l represented by equation

$$\mathbf{v} = \mathbf{v_0} + \lambda\ \mathbf{v_d}$$

or

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} X_0 \\ Y_0 \\ Z_0 \end{bmatrix} + \lambda \begin{bmatrix} a \\ b \\ c \end{bmatrix}, \tag{3.6}$$

which passes through the point $(X_0,\ Y_0,\ Z_0)$. In the above equation, $\mathbf{v_d} = (a\ b\ c)^T$ is the unit vector in the direction of the line and $\mathbf{v} = (x\ y\ z)^T$ is the vector pointing to any point on the straight line l. Then the plane Λ is defined by the equation

$AX + BY + CZ = D$ is sufficiently defined with the center of the lens and line l (provided line l does not pass through the center of the lens). Since the points $(0,\ 0,\ f)$ and $(X_0,\ Y_0,\ Z_0)$ are on the plane, we find that $D = Cf$ and therefore

$$AX_0 + BY_0 = C(f - Z_0)\ . \tag{3.7}$$

Now we can use $\mathbf{v_d}$, the direction of the line l, which is orthogonal to the normal vector of plane Λ, to formulate

$$Aa + Bb + Cc\ =\ 0\ . \tag{3.8}$$

Using Equations 3.7 and 3.8, we can express A and B in terms of C

$$\begin{aligned} A &= -C \begin{vmatrix} (z_0 - f) & y_0 \\ c & b \end{vmatrix} \Big/ \begin{vmatrix} x_0 & y_0 \\ a & b \end{vmatrix} \\ B &= -C \begin{vmatrix} x_0 & (z_0 - f) \\ a & c \end{vmatrix} \Big/ \begin{vmatrix} x_0 & y_0 \\ a & b \end{vmatrix} \end{aligned} \ . \tag{3.9}$$

Then the projected line l' in the image plane (z = 0) is given by $Ax + By = Cf$, which is equivalent to

$$\begin{vmatrix} (Z_0 - f) & Y_0 \\ c & b \end{vmatrix} x + \begin{vmatrix} X_0 & (Z_0 - f) \\ a & c \end{vmatrix} y \ = \ -f \begin{vmatrix} X_0 & Y_0 \\ a & b \end{vmatrix} . \tag{3.10}$$

From this it is obvious that l' is also a straight line. Hence straight lines map into straight lines.

3.2.2 Concept of Vanishing Points

The concept of vanishing points is closely related to the properties of the imaging transform described above in Subsection 3.2.1. At this point it is convenient to simplify the camera model used in Subsection 3.2.1 and define the following camera model: the origin of the system coincides with the center of the lens and the z-axis

with the optical axis. The image plane is defined by the equation $z = f$, where f is the focal length. Hence the plane is located in front of the center of the lens. Both models are exactly equivalent, except that the new model avoids inversion of the image.

Now we can examine the concept of vanishing points. Let l be a straight line through the base point $P_0 = (x_0, y_0, z_0)$ in three-dimensional space. The equation of l can be written as $\mathbf{v} = \mathbf{v_0} + \lambda\, \mathbf{v_d}$, which can be parameterized to

$$\begin{aligned} x &= x_0 + \lambda\, a \\ y &= y_0 + \lambda\, b \\ z &= z_0 + \lambda\, c \end{aligned} \quad . \tag{3.11}$$

We project any point on the straight line l into the image plane using the findings of Subsection 3.2.1. Using Equation 3.11, and considering the change of models by substituting $X_0 = x$, $Y_0 = y$ and $(Z_0 - f) = -z$ in Equation 3.1, we find that P_i, the point on the image plane, to be

$$\begin{aligned} x_i &= (f\, \tfrac{x_0 + \lambda\, a}{z_0 + \lambda\, c}) \\ y_i &= (f\, \tfrac{y_0 + \lambda\, b}{z_0 + \lambda\, c}) \\ z_i &= f \end{aligned} \quad . \tag{3.12}$$

As λ approaches infinity, we can neglect $P_0 = (x_0, y_0, z_0)$, the base point of the line. Then, with λ approaching infinity, P_i approaches the vanishing point V_l. One can define the vanishing point $V_l = (x_l,\ y_l,\ z_l)$ of a straight line l to be the projection of the line point with $\lambda \rightarrow \infty$ on the image plane, that is

$$\begin{aligned} x_l &= \lim_{\lambda\to\infty} f\,(\tfrac{x_0 + \lambda\, a}{z_0 + \lambda\, c}) = f\,(\tfrac{a}{c}) \\ y_l &= \lim_{\lambda\to\infty} f\,(\tfrac{y_0 + \lambda\, b}{z_0 + \lambda\, c}) = f\,(\tfrac{b}{c}) \\ z_l &= f \end{aligned} \quad . \tag{3.13}$$

Here the assumption has been made that $c \neq 0$, i.e., the straight line is not parallel to the image plane. If $c = 0$, then the vanishing point V_l does not exist (or V_l

is infinitely far away on the image plane). Since we neglect the base point of the line $P_0 = (x_0, y_0, z_0)$ and let $\lambda \rightarrow \infty$, the vector pointing in the direction of the vanishing point V_l is parallel to the direction of the straight line l. If we define the unit vector $\mathbf{v_l}$ pointing towards the vanishing point V_l as $\mathbf{v_l} = (\frac{x_l}{h} \quad \frac{y_l}{h} \quad \frac{z_l}{h})^T$ where $h = \sqrt{x_l^2 + y_l^2 + z_l^2}$, then the unit vector $\mathbf{v_l}$ is equivalent to the unit vector $\mathbf{v_d}$ which is the direction vector of the straight line l,

$$\mathbf{v_l} = \mathbf{v_d}\,. \tag{3.14}$$

For later reference it is useful to define two properties.

PROPERTY 2: The vanishing point V_l of a straight line l in three-dimensional space must lie on a line L on the image plane. Any segment of line l, projected onto the image plane, is part of line L [RK82b].

PROOF: Since the vanishing point V_L of a straight line l is defined as the projection of the line-point with $\lambda \rightarrow \infty$ on the image plane (see Equations 3.12 and 3.13), it is clear that the vanishing point has to lie on line L, the image of the line l, or on its extension [KK82].

PROPERTY 3: Let Ω be a set of parallel lines in three-dimensional space. Then all lines of Ω must have the same vanishing point V_Ω.

PROOF: All lines of the set Ω are parallel, and hence their direction unit vectors $\mathbf{v_d}$ are equivalent. Then using Equation 3.14 the unit vectors $\mathbf{v_\Omega}$, pointing towards the vanishing point V_Ω, are equivalent to the direction vectors $\mathbf{v_d}$.

All these findings about the concept of vanishing points are well understood and are partly described in [Bar83], [Bad74] and [CT90]. For a complete summary, refer to [KK82].

3.3 METHODS TO DETERMINE VANISHING POINTS

There are various methods to determine the vanishing points in a two dimensional image. These methods are more or less suitable depending on the requirements imposed by the application (precision, computation time etc.) and the heuristics available from *a priori* knowledge of the three dimensional scene. To provide an overview, we first briefly mention four methods to determine vanishing points, and in the following section we describe the cross-product method in details.

The objective is to find the principal vanishing points of a three dimensional scene, from a two-dimensional image. The three principal vanishing points of a room are defined as the vanishing points of the three room axes x, y and z (or of lines parallel to these axes). As shown in Subsection 3.2.2, parallel lines in a three-dimensional scene have the same vanishing point in the image plane (Property 3). The vanishing point of a three-dimensional line lies on the projection of that line in the image plane or on its extension (Property 2). Hence, given a two-dimensional image with detected line segments, we can determine vanishing points by finding the intersections of line segments (or their extensions). Note that not every intersection between lines is a vanishing point, and hence we have to distinguish between "accidental" intersections of line segments and vanishing points. To solve this problem we have to use the heuristics that a principal vanishing point is formed by the intersection of many line segments. This assumption is true for many man-made indoor and outdoor objects [SK80].

Kender [Ken79] presents a two step approach to determine vanishing points. The first step maps line segments with a common vanishing point into circles that pass through the origin. The vanishing point itself appears at the point on the circle

that is most distant from the origin. Determining which lines are associated with the same vanishing point involves finding circles in this space. This is a problem which requires a complicated search. Therefore in the second step, Kender maps the lines using a Hough like parameterization. Thus, the problem of determining vanishing points reduces to finding lines in the space.

Barnard [Bar83] determines the plane Λ which contains the lens center and the line segment in the image plane. The great circle which results by intersecting the plane Λ with the Gaussian sphere is traced out in an angular (azimuth, elevation) parameterization of the Gaussian sphere. After all great circles have been traced, vanishing points appear in the parameterized space as elements with a high occurrence rate, i.e., many great circles representing various lines pass through this element on the Gaussian sphere. The primary problem that Barnard points out is that using elevation and azimuth to divide the sphere causes sphere elements to be non-uniform in size. L. Quan and R. Mohr [QM89] suggest a hierarchical search to find vanishing points on the Gaussian sphere. This improves the computational efficiency of the method.

Badler [Bad74] suggests a method based on cross products. The vector towards an intersection is computed with the cross product of vectors to line segments and the cross product of normal vectors of planes associated with the line segments. We refer to this method as the cross-product method. The work of Magee and Aggarwal [MA84] extends this cross-product method to address the issues of working in calibrated as well as non-calibrated imaging systems. A more detailed description of that approach is given in Subsection 3.4.

Shafer *et al.* [KK82] deal with vanishing point issues as they relate to gradient space under orthographic and perspective projections. Their derivations show that lines, which are parallel in three-dimensional space, have a unique vanish-

ing point that is determinable by using gradient operations. Relationships among points, lines, and planes in the gradient space are also presented.

Weiss *et al.* [WH88] provide an error analysis on the detection of vanishing points, vanishing lines and the orientation of a plane associated with a vanishing line.

3.4 CROSS-PRODUCT METHOD

The Cross-Product Method has been described by Magee and Aggarwal [MA84] and is based on a previous work by Barnard [Bar83]. The approach uses the Gaussian sphere to represent points on the image plane (Figure 3.2). Any point (x, y) on the image plane ($z = f$) can be mapped onto the Gaussian sphere (centered at the origin) using $\mathbf{p}/|\mathbf{p}|$, where $\mathbf{p} = (x \ y \ f)^T$. Any vector from the origin to a point, mapped on the Gaussian sphere, can be uniquely described in terms of azimuth α (the angle between the projection of the vector on the xy-plane and the x-axis) and the elevation β (the angle between the vector and the xy-plane). There are two main reasons for using the Gaussian Sphere. As described in Subsection 3.2.2 vanishing points can be finite and infinite. Infinite vanishing points on the image plane are difficult to handle in an algorithm (due to computation as well as representation). The Gaussian sphere accommodates both finite and infinite cases of line intersections or vanishing points. Moreover, the mapping onto the Gaussian sphere converts the distance between two points in the image plane into an angle. This is a much more natural way of looking at the three dimensional space and has advantages when defining the clustering criterion for intersections (Subsection 3.5.2).

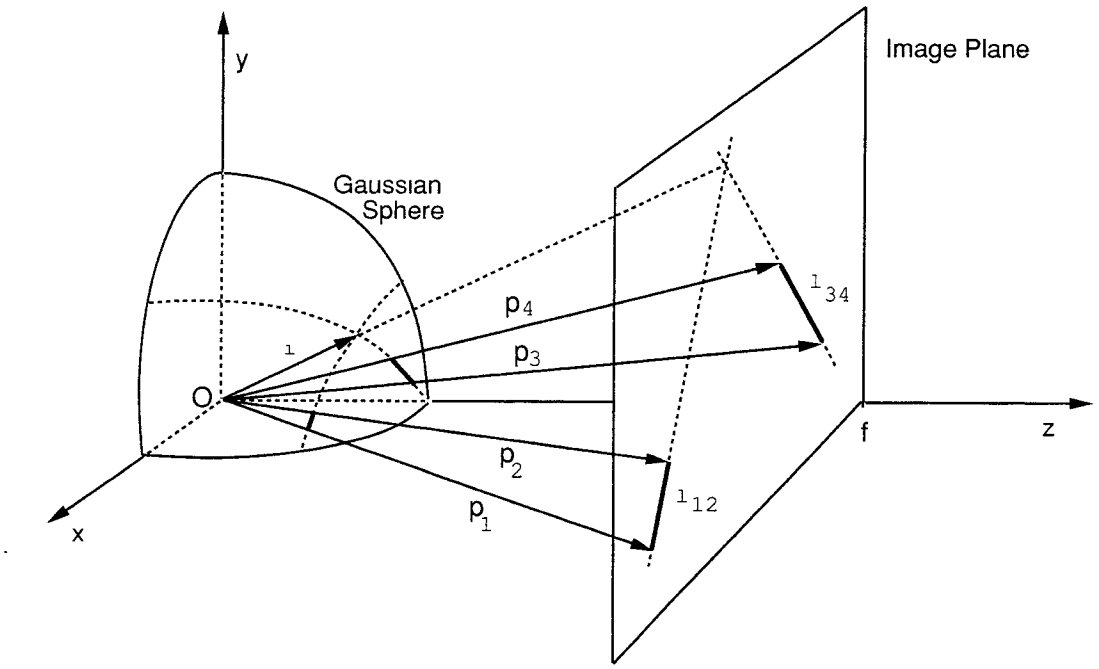

Figure 3.2: *Gaussian sphere and computation of the vector pointing towards the intersection of two line segments (f denotes the focal length)*

3.4.1 Computing the Directions of Intersections

Magee and Aggarwal [MA84] suggest a computationally efficient way to find the intersections of line segments. Instead of using Barnard's great circles on a Gaussian sphere [Bar83], they find the vector pointing towards the line intersection using three cross products. Assuming two line segments l_{12} and l_{34} on the image plane (Figure 3.3), we can define four vectors to the start and end points of the line segments as

$$\mathbf{p_1} = \begin{pmatrix} x_1 \\ y_1 \\ f \end{pmatrix}, \mathbf{p_2} = \begin{pmatrix} x_2 \\ y_2 \\ f \end{pmatrix}, \mathbf{p_3} = \begin{pmatrix} x_3 \\ y_3 \\ f \end{pmatrix}, \textit{and } \mathbf{p_4} = \begin{pmatrix} x_4 \\ y_4 \\ f \end{pmatrix}. \qquad (3.15)$$

The vectors $\mathbf{p_1}$, $\mathbf{p_2}$ and $\mathbf{p_3}$, $\mathbf{p_4}$ are pointing towards the line l_{12} and line l_{34}, respectively. With these definitions we can find the unit normals, to the planes defined by two vectors, to a line segment. Using the vectors forming the planes, the unit

normals $\mathbf{n_{12}}$ and $\mathbf{n_{34}}$ can be computed as

$$\mathbf{n_{12}} = \frac{\mathbf{p_1} \times \mathbf{p_2}}{|\mathbf{p_1} \times \mathbf{p_2}|}, \tag{3.16}$$

$$\mathbf{n_{34}} = \frac{\mathbf{p_3} \times \mathbf{p_4}}{|\mathbf{p_3} \times \mathbf{p_4}|}. \tag{3.17}$$

Accordingly, the vector $\bar{\mathbf{i}}$ along the intersection of the two planes can be computed with the cross product of the unit normals $\mathbf{n_{12}}$ and $\mathbf{n_{34}}$.

$$\bar{\mathbf{i}} = \begin{pmatrix} \overline{x_i} \\ \overline{y_i} \\ \overline{z_i} \end{pmatrix} = \mathbf{n_{12}} \times \mathbf{n_{34}} \tag{3.18}$$

or with $\bar{\mathbf{i}}$ mapped on the Gaussian sphere,

$$\mathbf{i} = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} = \frac{\mathbf{n_{12}} \times \mathbf{n_{34}}}{|\mathbf{n_{12}} \times \mathbf{n_{34}}|} = \frac{1}{w} \begin{pmatrix} \overline{x_i} \\ \overline{y_i} \\ \overline{z_i} \end{pmatrix} \tag{3.19}$$

where

$$w = \sqrt{\overline{x_i}^2 + \overline{y_i}^2 + \overline{z_i}^2} \tag{3.20}$$

Combining Equations 3.16, 3.17 and 3.18 we can compute the components $\overline{x_i}$, $\overline{y_i}$ and $\overline{z_i}$ to be

$$\overline{x_i} = (x_2 - x_1)(x_3 y_4 - x_4 y_3) - (x_4 - x_3)(x_1 y_2 - x_2 y_1) \tag{3.21}$$

$$\overline{y_i} = (y_3 - y_4)(x_1 y_2 - x_2 y_1) - (y_1 - y_2)(x_3 y_4 - x_4 y_3) \tag{3.22}$$

$$\overline{z_i} = (x_4 - x_3)(y_1 - y_2) - (x_2 - x_1)(y_3 - y_4) \tag{3.23}$$

If the line segments or their extensions have an intersection in the image plane, the vector $\mathbf{i}$ is directed towards the point of intersection (see Figure 3.3 for illustration). If the lines are exactly parallel in the image plane then $\mathbf{i} = \mathbf{0}$. Due to the properties of cross products, vector $\mathbf{i}$ might point in the opposite direction (with $z_i < 0$), rather than towards the point of intersection. The algorithm which processes the intersection has to be prepared for this case.

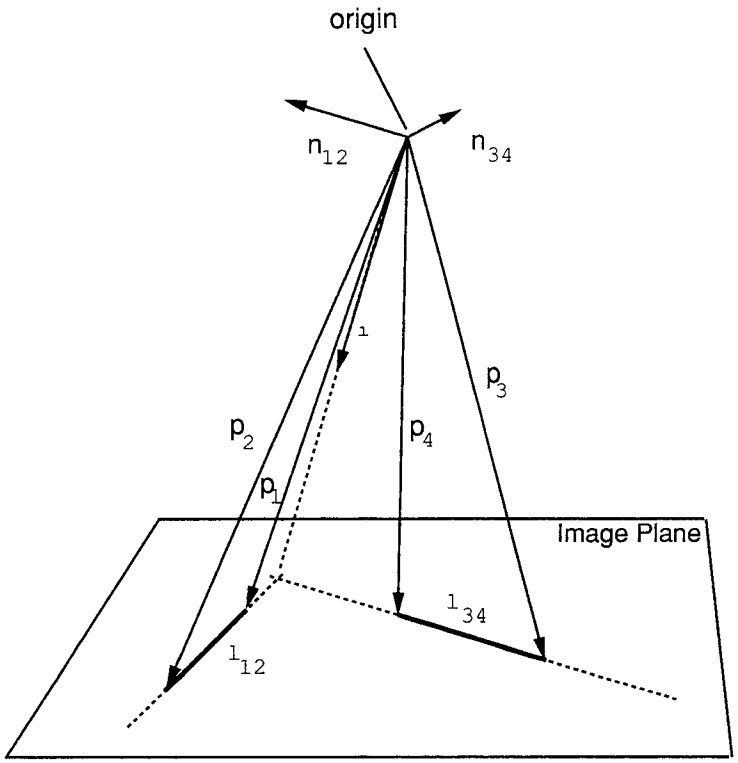

Figure 3.3: *Computation of the vector towards the intersection of the line segments*

3.4.2 Processing the Intersections

Assuming an image with the line segments detected, we can summarize the cross-product method in four steps [MA84].

STEP 1: Given the line segments and the vectors pointing towards their start and end points, we can compute the vector $\mathbf{i}$ pointing towards the intersection of the line segments using three cross products (Subsection 3.4.1). To ensure that all vectors $\mathbf{i}$ point to the hemisphere of the Gaussian sphere where $z_i > 0$, the z_i component of $\mathbf{i}$ has to be examined. If z_i is negative we have to negate each component of $\mathbf{i}$, which results in $-\mathbf{i}$ (collinear to $\mathbf{i}$).

STEP 2: Since we do not know which line segments form a common vanishing point, we have to compute all intersections between all possible pairs of line

segments. These are $N(N-1)/2$ intersections (where N denotes the number of line segments). This can be a very large number. In order to limit the number of intersections, we do not accept a vector **i** if it points towards the interior of either of the two line segments that are used to compute vector **i**. In general, vanishing points cannot lie between the start and the end-point of a line segment since vanishing points are defined with λ approaching infinity (Equation 3.13). Therefore this constraint filters out intersections which are not true vanishing points of three dimensional parallel lines. There are other filtering constraints which will be discussed later.

STEP 3: The accepted intersection vectors **i** are represented with azimuth α and elevation β, which are defined by

$$\alpha = arctan\left(\frac{y_i}{x_i}\right) , \tag{3.24}$$

$$\beta = arctan\left(\frac{z_i}{\sqrt{(x_i)^2+(y_i)^2}}\right) . \tag{3.25}$$

The line segments that intersect at a particular (α, β) pair are associated with that (α, β) combination. If it is the first time the (α, β) pair is encountered, a new association for (α, β) is created. If there is a previous association, then just add this pair of line segments onto the list of line segments for that (α, β). Note that "associating" does not involve quantization of angles. The actual values of (α, β) are kept exactly as they are (unlike Barnard's approach [Bar83]).

STEP 4: After all pairs of line segments have been processed, we have to cluster the intersections, i.e., we have to decide which of the intersections are close enough (using angular distance), such that they could contribute to the same vanishing point. The angular distance between two intersections represented by α_1, β_1 and α_2, β_2 can be computed using

$$\delta = arccos[cos(\pi/2-\beta_1)\, cos(\pi/2-\beta_2) +$$

$$sin(\pi/2 - \beta_1)\ sin(\pi/2 - \beta_2)\ cos(\alpha_1 - \alpha_2)] \tag{3.26}$$

(see [MA84] for details). This is done by comparing the first association against the rest. If the number of associations (α, β), within the angular distance δ, is larger than some threshold t, then those associations are tagged as belonging to the same vanishing point. The algorithm then proceeds with finding other vanishing points using only the untagged associations. Hence the algorithm accepts an (α, β) pair as a vanishing point if it finds more associations than a threshold t within an angular distance of δ. This implies the heuristics, that the image has more intersections within distance δ from a vanishing point than "accidental" intersections within a distance δ.

In addition to describing this algorithm, Magee and Aggarwal [MA84] prove that this method works even if the focal length f is not known. Substituting f with a positive number s in the vectors defined by Equation 3.15 will still yield the same intersecting points in the image. The lines contributing to one vanishing point can still be grouped. Hence costly camera calibration procedures can be avoided. Note that the algorithm is sensitive to very large as well as very small values of s. A reasonable value for s is the image width and height in pixels. It is obvious that applying the algorithm in an uncalibrated system can cluster the lines contributing to one vanishing point, but it does not determine the true direction of the vanishing point from the center of the lens in the three dimensional space.

3.5 MODIFIED CROSS-PRODUCT METHOD

The cross-product method is appropriate for images with strong, distinguished vanishing points and only few accidental intersections which do not belong to any

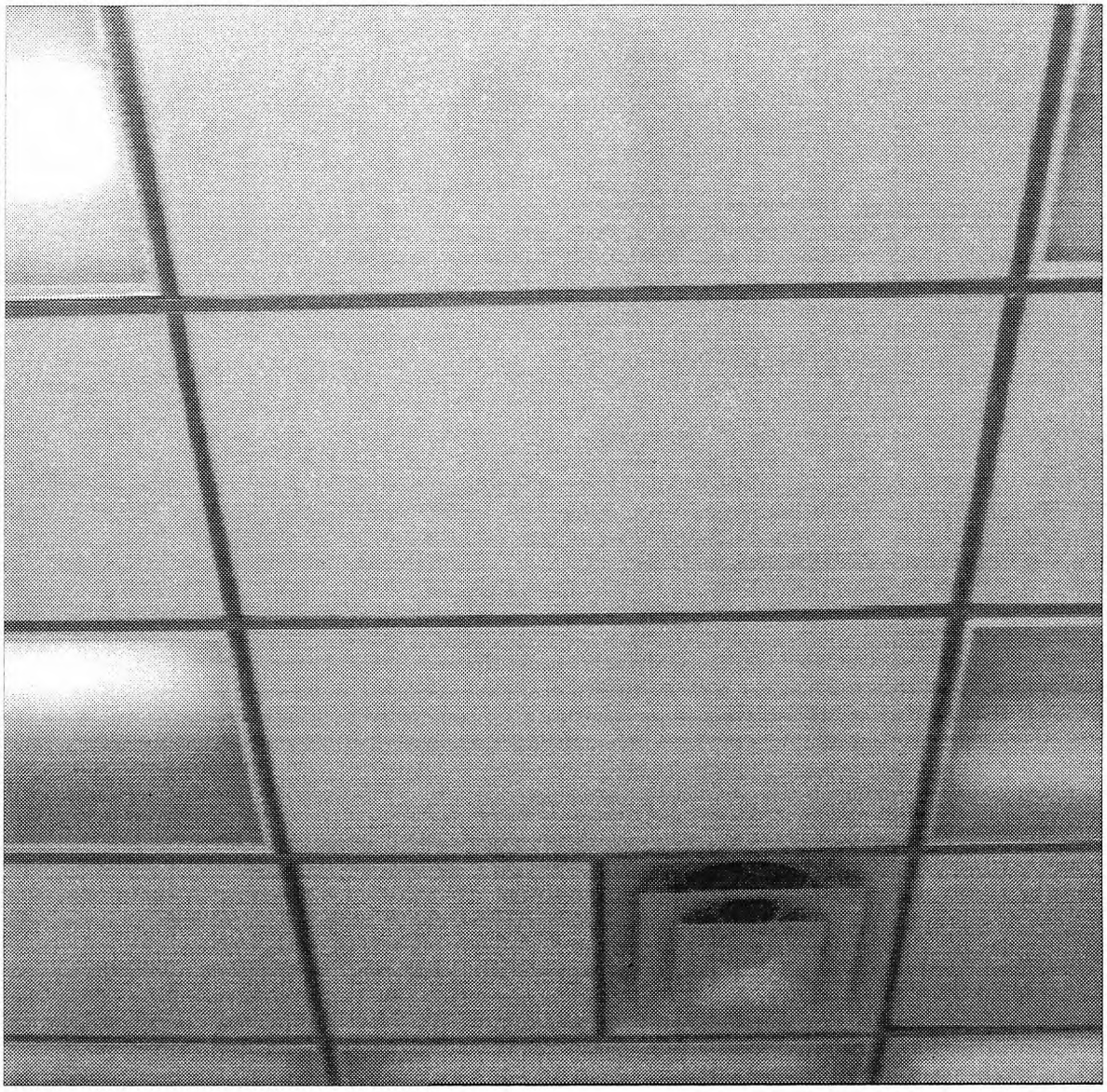

Figure 3.4: *Image of a common ceiling pattern under perspective distortion.*

vanishing point. In general, this is not true for real indoor or outdoor images. In particular in an indoor environment we cannot assume that the vanishing point we want to determine accommodates more than a certain number of lines and we are confronted with many accidental intersections, which do not contribute to a vanishing point. In addition, in images of a typical indoor ceiling grid, we get many parallel line segments very close to each other (see Figure 3.4). Naturally, the intersections of those line segments are not useful since they are very inaccurate (see Subsection 3.5.2). In order to use the cross-product method in an indoor environment with the requirements mentioned above, we modify the following features of the algorithm:

- Additional constraints for filtering line segments, intersections and (α, β) associations at different stages of the algorithm.
- Different representation of (α, β) associations and different clustering algorithm.
- New approach to distinguish between determined vanishing points.

The last item addresses an issue which is not included in the cross-product method. If we want to use the vanishing points for specific applications, we need additional constraints to select a particular vanishing point from all vanishing points determined by the cross-product method. One of the most significant changes to the original approach of Magee and Aggarwal [MA84] is the different representation of the (α, β) associations. We define a two-dimensional array (accumulator) with 128×32 elements. Each accumulator element represented by (a, b) contains a list of line segments and the count of those line segments. We quantize the occurring (α, β) association in the accumulator such that the values of α (0° to 360°) are mapped into a (128 sections) and the values of β (0° to 90°) are mapped into b (32

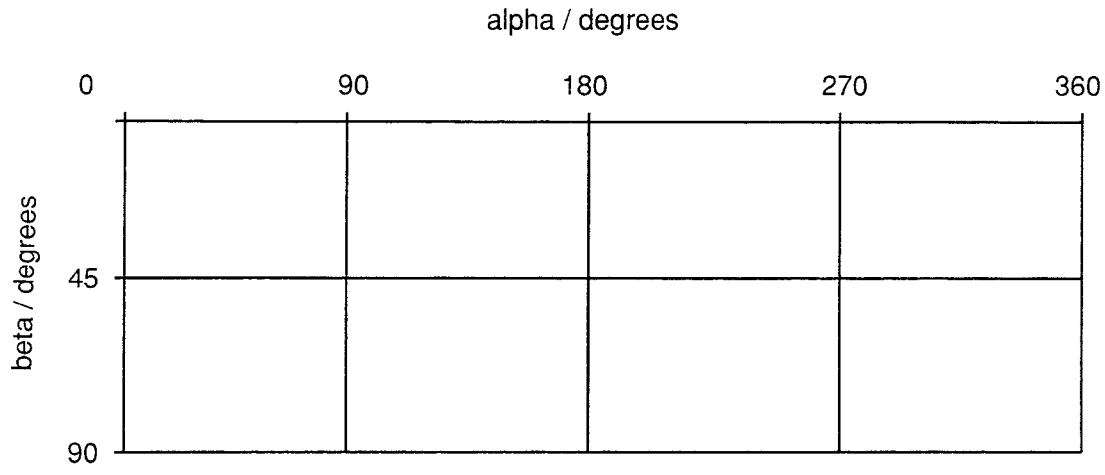

Figure 3.5: *Two dimensional accumulator array representing the Gaussian sphere.*

sections). See Figure 3.5 for illustration. Thus we define a grid on the Gaussian sphere with an angular width and height of 2.81 degrees since

$$360° / 128 = 90° / 32 = 1.28°. \tag{3.27}$$

In general, the areas on the Gaussian sphere that the accumulator elements correspond to are not equal. Other forms of representation, such as tessellated regular polyhedra, might be better suited to represent the sphere, but they are rather complicated to implement. Note that for a constant azimuth a, the areas on the Gaussian Sphere are equal. The algorithms of [Bar83] and [QM89] use the same representation.

3.5.1 Implementation of modified cross-product method

The cross-product method has been modified according to the different requirements and constraints in an indoor environment. It can be summarized in five steps.

STEP 1: As before, we can compute vector $\mathbf{i}$, pointing towards the intersection of the line segments, with three cross products (Subsection 3.4.1). Here we add two constraints:

- Only consider those line segments which are longer than a minimum length l_{min}.
- Do not intersect line segments that are approximately parallel and very close to each other.

Both constraints are introduced to limit the number of intersections and to increase the accuracy. The location of intersections computed with short line segments are not accurate (see Subsection 3.5.2). The same is true for line segments which are approximately parallel and close to each other. The second constraint is implemented using the ρ, θ representation of lines. If ρ_1 and θ_1 of line segment l_1 both differ from ρ_2 and θ_2 of line segment l_2 within certain limits, then we do not compute the intersection of that pair of line segments. As in the original cross-product method, we have to ensure that all vectors $\mathbf{i}$ pointing towards intersections, point to the hemisphere of the Gaussian sphere where $z > 0$.

STEP 2: Indoor scenes usually produce images with many "accidental" intersections, i.e., intersections that are not associated with a vanishing point. Figure 3.6 displays the accumulator with all computed intersections (the algorithm does not filter intersections). A large number of accumulator elements contain intersections (white elements) and it is clear that the detection of vanishing points is possibly ambiguous. This illustrates the importance of intersection filtering. For the indoor environment the constraint for intersection filtering used in the original cross-product method falls short of filtering capabilities. It is not sufficient to check whether an intersection is lying inside either one of the line segments which

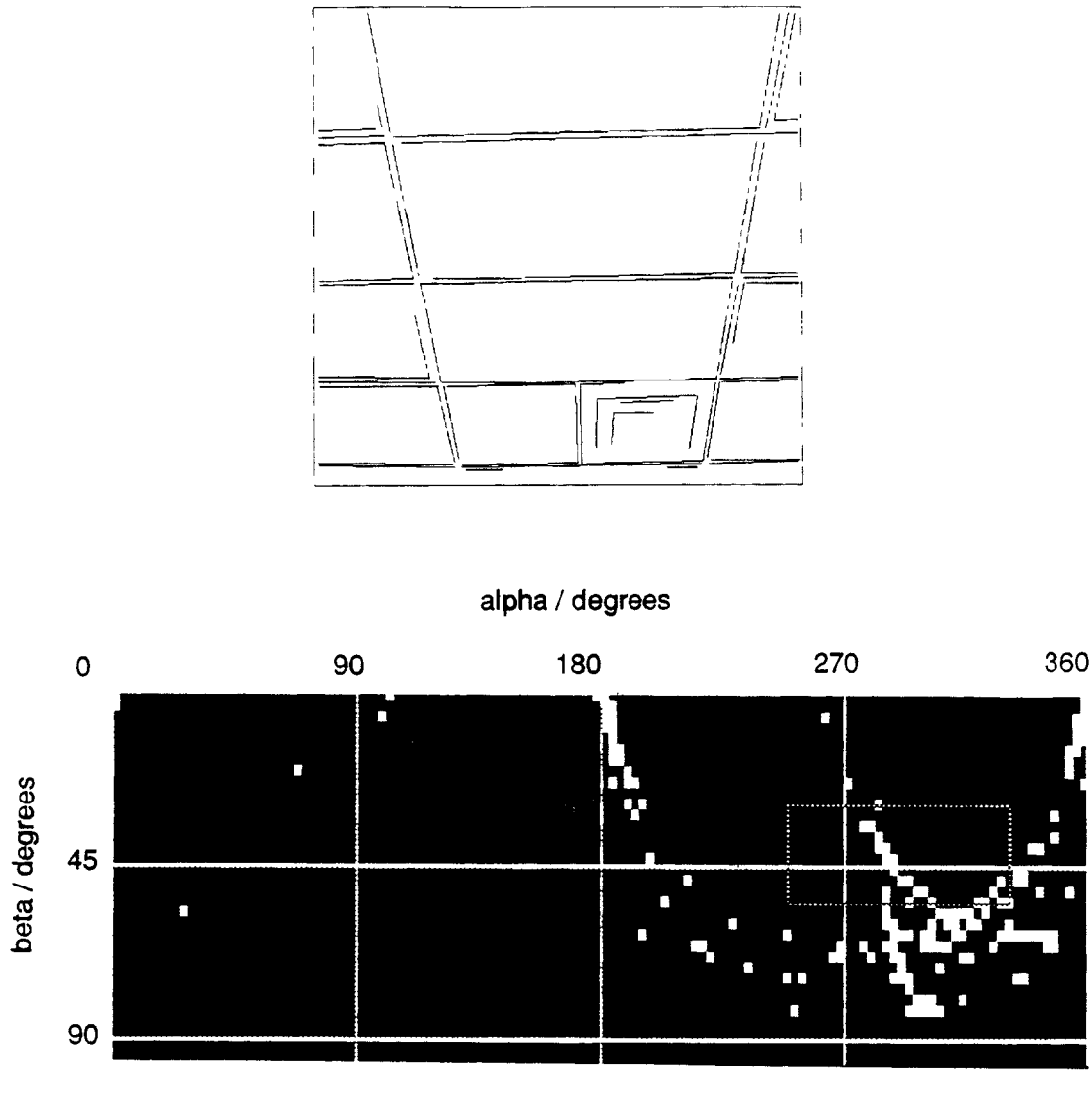

Figure 3.6: *Line segments in ceiling pattern (top image) and intersections of line segments (bottom image) in accumulator representation (no intersection filtering applied).*

are forming that intersection. Additional constraints are necessary, for example:

- Do not accept intersections that are interior to *extended* line segments. Line segments can be extended on both sides of the segments by a constant factor.
- Do not accept intersections that are inside the image plane (or the image plane extended by a constant factor).

Experiments show that the constraint using extended lines is sufficient for ordinary indoor scenes. In case of images from an indoor ceiling grid (most line segments in two directional groups) it seems to be practical to accept only those intersections which are outside of the actual image plane with image width and image height in pixels. Since all line segments are oriented in two directional groups most of the "accidental" intersections are inside the image plane. Note that this last constraint can only be used for images which have their vanishing point outside of the image plane. This may not be true for many indoor scenes. Remember that all these constraints are intended to increase the computational efficiency. If there are no time requirements and there is enough system memory available, this step can be ignored.

STEP 3: Equations 3.24 and 3.25 are used to compute the angles α and β of the accepted intersection represented by $\mathbf{i} = (x_i \ y_i \ z_i)^T$. Then we find the accumulator grid element (a_n, b_n) corresponding to the intersection (α, β) on the Gaussian sphere. The line segments associated with the intersection (α, β), are added to the line list of the accumulator element (a_n, b_n) (if they are not already there). According to the number of line segments actually added to the line list, the line count of the accumulator element (a_n, b_n) is increased. After all intersections are processed, the accumulator provides the number of line segments that intersect within each grid element on the Gaussian sphere.

STEP 4: Given the line count in each accumulator element we have to cluster adjacent accumulator elements which have non-zero line count. Adjacent accumulator elements that have a non-zero line count most likely contribute to the same vanishing point. The clustering algorithm scans the accumulator twice (row by row, top to bottom) and labels connected elements that have non-zero line count (clusters), with the same label [RK82a, page 241]. In the first scan for each non-zero accumulator element (x, y) four neighbors u_0 to u_3 are examined.

$$\begin{array}{cc} u_1 & u_2 \quad u_3 \\ u_0 & (x, y) \end{array}$$

Based on the labels of u_0 to u_3 the algorithm assigns a label to the accumulator element (x, y). Three cases have to be considered. First, if there is no label found in u_0 to u_3, (x, y) is assigned a new label. Second, if there is only one label found in u_0 to u_3, this label is assigned to (x, y). Third, if there are two different labels L_0 and L_1 found in u_0 to u_3, label L_0 is assigned to (x, y) and the fact that L_0 is equivalent to L_1 is recorded. Once the first scan is completed, pairs of equivalent labels are sorted into equivalence classes, and one label is chosen to represent each class. The second scan replaces each label in the accumulator with the representative of its class. After the second scan is completed all accumulator elements of one cluster are labeled with one unique label.

Now that we have found the clusters of intersections we can approximate the center of the intersections with weighted averaging. With the line count as the weight for each accumulator element, the center of a cluster $(\alpha_{center}, \beta_{center})$ can be computed with

$$\alpha_{center} = \frac{360^\circ \sum_{i=1}^{K} n_i a_i}{128 \sum_{i=1}^{K} n_i} \tag{3.28}$$

$$\beta_{center} = \frac{90^\circ \sum_{i=1}^{K} n_i b_i}{32 \sum_{i=1}^{K} n_i} \tag{3.29}$$

where n_i represents the line count, a_i and b_i indicate the position of the element in the accumulator and K is the number of accumulator elements in that cluster. The constant factors only transform the accumulator representation into angles in degrees.

Hence this modified cross-product method provides the direction of all vanishing point candidates uniquely determined with azimuth α_{center} and elevation β_{center}. Now the desired vanishing point has to be selected from the group of candidates. Here we have to rely on heuristics and *a priori* knowledge from the images. For most applications it is sufficient to use the highest number of intersecting line segments as a selection criterion. Applying this criterion we select the vanishing point candidate with the highest sum of line counts ($\sum_{i=1}^{K} n_i$) within its cluster as the vanishing point. Depending on the application, this might not yield the desired results. We might want to detect a vanishing point which does not necessarily have the highest number of line segments associated with its location. Provided that there exists some *a priori* knowledge of the approximate location of the vanishing point, it is possible to define a window in the accumulator describing the area where the vanishing point is expected to be. If we apply the above constraint only inside the window, we can select a vanishing point which has the highest line count within the window but not necessarily within the entire accumulator.

3.5.2 Error Considerations

The entire process of determining the vanishing points introduces many different sources of error. The errors that resulted in obtaining the line segments can be summarized in three groups.

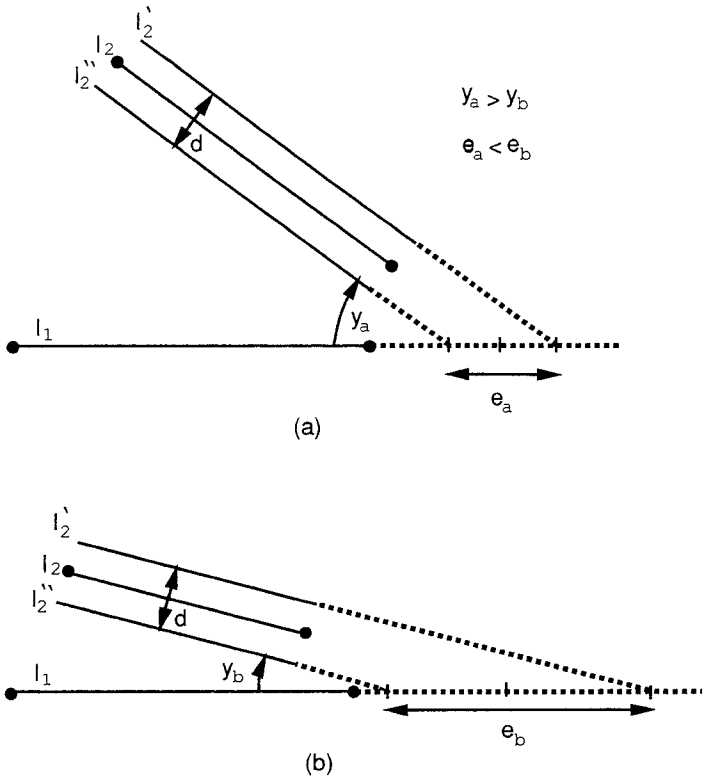

Figure 3.7: *Shift Error of line intersections: Assuming an error bound d parallel the line l_2, then the error of the intersection is a function of the angle y between the lines.*

- Errors related to the optical system: There is perspective distortion associated with the imaging transform (Subsection 3.2.1). Pincushion and barrel distortion [RK82b, page 26] result from a slight change of scale in the image. In pincushion distortion, the scale of the image increases with distance from the center of the image. Barrel distortion is related to a scale decrease with the distance from the center of the image.

- Errors related to the digitization of the image in the image plane: Here we have to consider the spacial quantization in discrete pixels and the quantization of the detected light in discrete gray values as a source of error.

- Errors related to image processing: Errors result from edge detection and line fitting.

The above errors primarily result from the extraction of line segments. Discrepancy in the extracted line segments will inevitably cause errors in the intersection points of the line segments, and thus errors in the vanishing points. There are two types of errors, shift and turn errors. Figure 3.7 shows the shift errors associated with line l_2. Line l_2 can be located anywhere between l_2' and l_2'' where its direction is parallel to l_2. Figure 3.7 illustrates that shift errors are a function of the angle between the line segments. The larger the angle y between the line segments l_1 and l_2, the smaller is the error in detecting the intersection. This implies that stronger perspective distortion of line segments in three dimensional space reduces the error in detecting the intersection. Conversely, approximately parallel line segments produce inaccurate intersections.

Errors that cause an end point of a line segment to move to a nearby pixel location such that the line segment is turned, is referred to as turn errors. Assume that errors in an extracted line segment can cause its end points to be located

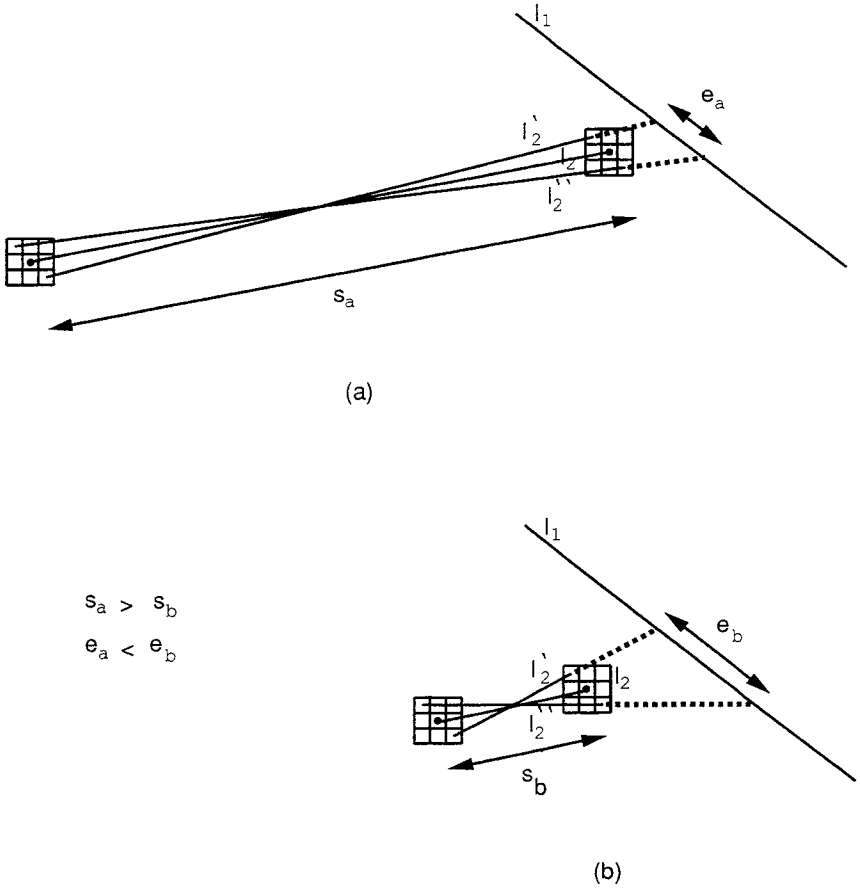

Figure 3.8: *Turn error of line intersections: Assuming a turn of the line l_2 within a eight pixel neighborhood, then the error of the intersection is a function of the length s of line l_2 and the distance between the lines.*

within its 8 pixel neighborhood as shown in Figure 3.8. Lines l_2'' and $l_2''{}'$ show the worst case turn of line l_2. Again, for simplicity l_1 is assumed to have no error. The turn error is a function of the length of the line segment. The longer s (the length of line segment l_2) is, the smaller is the error e. Also, the farther the separation of the line segments l_2 and l_1 (Figure 3.8(b)), the larger is the error e. These are the reasons why intersections of short line segments are not accurate. Errors in both line segments, l_1 and l_2 will cause the intersecting point to be located within an area of error.

The cross-product method maps the lines segments onto the Gaussian sphere. The error associated with the line segments can be mapped similarly. Therefore, the intersection error on the Gaussian sphere is the same as described above. Weiss *et al.* [NR88] provide a complete error analysis on the Gaussian sphere. They assume an error associated with the detection of line segments. Then they propagate the errors of vanishing points, vanishing lines and surface orientation based on the error of the line segments.

3.6 RESULTS AND CONCLUSION

The modified cross-product method described above has been applied to images of indoor scenes. Figure 3.4 shows the original image of a room ceiling with a common grid pattern, texture and lamps. The edges and lines have been detected by the combined Laplacian and Sobel operators and the line fitting algorithm described in Chapter 2. The detected lines are shown in Figure 3.9 (top image). The ceiling grid contains two groups of detected lines. They are referred to as horizontal and vertical line group. The lines of each group should form one common vanishing point. The modified cross-product method is applied to the detected lines. Since the im-

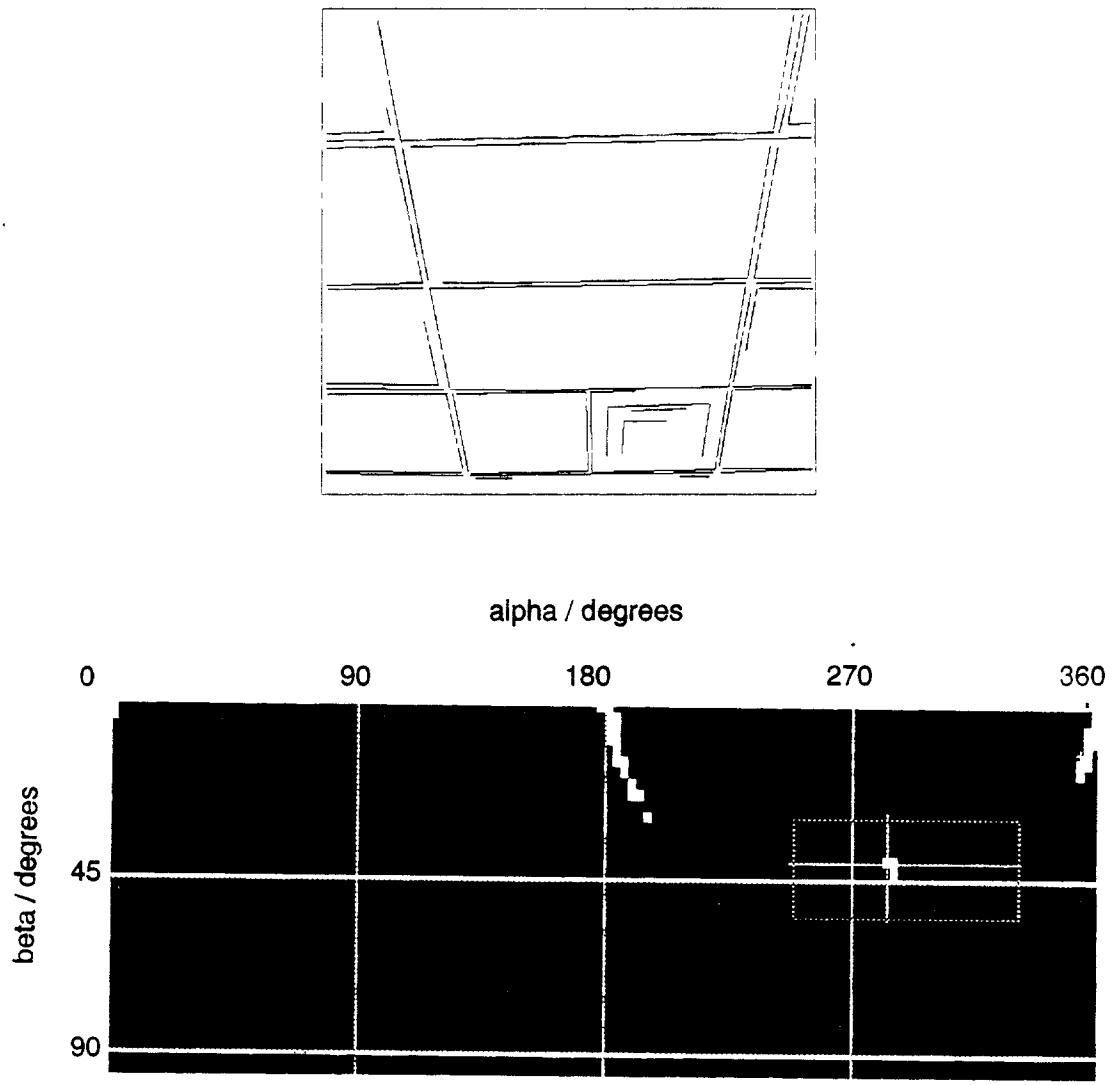

Figure 3.9: *Line segments in ceiling pattern (top image) and intersections of line segments (bottom image) in accumulator representation (intersection filtering applied).*

age has a large number of "accidental" intersections within or close to the image plane (see Figure 3.6), the algorithm eliminates those intersections that are located within the image plane extended by a constant factor. Figure 3.9 (bottom image) shows the resulting intersection detected by the modified cross-product method. It displays the two-dimensional accumulator array representing the Gaussian sphere with (α, β) elements. White accumulator elements indicate that there are intersections located within that area of the Gaussian sphere. Black accumulator elements do not contain any intersections. Three large clusters are detected. The smallest cluster around $(\alpha = 285°, \beta = 40°)$ results from the vertical line group. The other two clusters on the right and the left of the first one result from the horizontal line group. Since these lines are almost parallel, their angle of intersection is very small, and thus the error of intersection is very large (see Subsection 3.5.2). The intersections of that line group are not located within one accumulator element only, they are spread over a large region. Since there are more horizontal lines than vertical lines detected, the clusters resulting from the horizontal lines have a higher line count. If we want to select the cluster of the vertical lines as a vanishing point, we have to use a window (provided the approximate location of the vanishing point is known). The window concept is described in Subsection 3.5.1. Figure 3.9 (bottom image) shows the window used to select the cluster of the horizontal lines as the vanishing point.

This shows clearly that the modifications made to the original cross-product method are necessary to enable the algorithm to work in an indoor environment. These results illustrate the advantages and limitations of vanishing points as well as the usefulness of the modified cross-product method.

Chapter 4

NAVIGATION

4.1 INTRODUCTION

In general, any approach to robot navigation that uses any type of sensors requires a model of the environment of the robot. In other words, if we want to obtain information about the environment of the robot via sensors, we first have to define that environment. Hence we have to make certain assumptions. For example, Elfes [Elf87], [ME89] chooses a probabilistic approach using sonar sensors. This implies that there are objects in the environment of the robot that reflect sonar signals. For any navigation method using sensors, the model of the environment has to be defined as general as possible to include a broad class of objects.

Recent research work on autonomous mobile robots shows that current development is still far from achieving human-like performance. The current so-called intelligent and autonomous robotic systems are still very primitive as compared to human sensing and reaction capabilities. Therefore it is still interesting to investigate little facets of the overall task separately. In this work, we look into vanishing points and their usefulness in robot navigation. We are not concerned with functions or "behaviors" [Bro89] like obstacle avoidance or object detection. The focus

here is on guiding a mobile robot in a specific direction using the vanishing points of lines in the robot environment. The robot environment is mostly assumed to be an indoor environment where the lines can be ceiling or floor patterns, but the technique is not restricted to that environment. An attempt is made to define a model for the robot environment which contains a minimum of constraints necessary for navigation based on vanishing points (Section 4.2).

4.2 DEFINITION OF THE GENERIC MODEL

As mentioned above, robot navigation with sensors requires a model of the environment of the robot. Binford and Kriegman [KB88] suggest the definition of a generic model. A *generic model* is a single model that describes a large class of objects. This implies that the generic model of an environment is as general as possible and uses a minimum of constraints on the environment. A generic model should not represent a particular environment, instead it should describe a large class of environments. This ensures that the sensoring of the navigation method is applicable to many different environments.

Binford and Kriegman [KB88] define a generic model for buildings and indoor environments. The functional constraint on the generic building model asserts that the primary occupants of buildings are people, which determines the scale of a building. The minimum dimensions of building elements, like doors, ceilings and walls, are constrained by the size of a typical large person. Economics is considered to place the limit on the maximum dimensions of buildings. It is often too expensive to make something larger than necessary. One of the primary physical constraints on buildings is gravity. A floor which is planar for motion efficiency, is generally normal to the gravity vector. In buildings with multiple stories, the floors

constrain the ceilings. Hence they are also normal to the gravity vector.

The generic model for the environment of a robot guided by vanishing points can be defined with three constraints:

- A minimum of two parallel, straight lines are detected by the camera.
- The direction of the lines is known with respect to the robot environment.
- The direction of the lines is not parallel to the image plane of the camera.

The underlying principles which make these constraints sufficient are explained later (Section 4.4). This generic model describes any environment which provides parallel, straight lines with any known direction (not parallel to image plane). This covers a wide range of different indoor and outdoor environments. In the following we use the generic model in an indoor environment. In an indoor environment. any pattern with straight lines, like the ceiling grid, the wall paper, or the floor tiles can be used for navigation with vanishing points. There are no assumptions necessary on the orientation of the surfaces containing the lines (ceilings, floors, walls). Only the direction of the lines with respect to the environment has to be known.

4.3 COORDINATE TRANSFORM

Before we explore the properties of vanishing points in navigation, we have to define the relationship between the objects detected by the camera and the environment of the robot. We can define two coordinate systems (three dimensional). The first one represents a room in an indoor environment (room system), and the second one represents the space the camera points to (camera system). The origin of the camera system can be located at any point in the room system. The two coordinate

systems can be rotated with respect to each other. If we want to find the relation between the room system and the camera system, we have to consider translation and rotation (if both systems are assumed of equal scale). In general, translation and rotation can be expressed as matrices. The translation matrix $\mathbf{T}$ is defined by

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -T_x & -T_y & -T_z & 1 \end{bmatrix}, \tag{4.1}$$

where T_x, T_y and T_z represent the translation components on the three axes of the room system. Rotation about the three axes x, y and z is defined by $\Re_\eta$, $\Re_\mu$ and $\Re_\theta$, respectively. Hence a rotation of an angle η, about the x axis is defined by

$$\Re_\eta = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\eta & -\sin\eta & 0 \\ 0 & \sin\eta & \cos\eta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{4.2}$$

a rotation of angle μ about the y axis is defined by

$$\Re_\mu = \begin{bmatrix} \cos\mu & 0 & \sin\mu & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\mu & 0 & \cos\mu & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{4.3}$$

and a rotation of angle θ, about the z axis is defined by

$$\Re_\theta = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{4.4}$$

(see [RK82a] and [GW87] for the derivation of the matrices). Let the room system be described with the x', y' and z'-axes and the camera system be represented by the x, y and z-axes (Figure 4.1). Let $\mathbf{p'_h} = (kx' , ky' , kz' , k)^T$ be the vector representing the point P' in the room system, and let $\mathbf{p}$ be the corresponding point and $\mathbf{p_h} = (kx , ky , kz , k)^T$ be the vector in the camera system. The vectors are give in homogeneous coordinates, where k is an arbitrary non-zero constant. Then

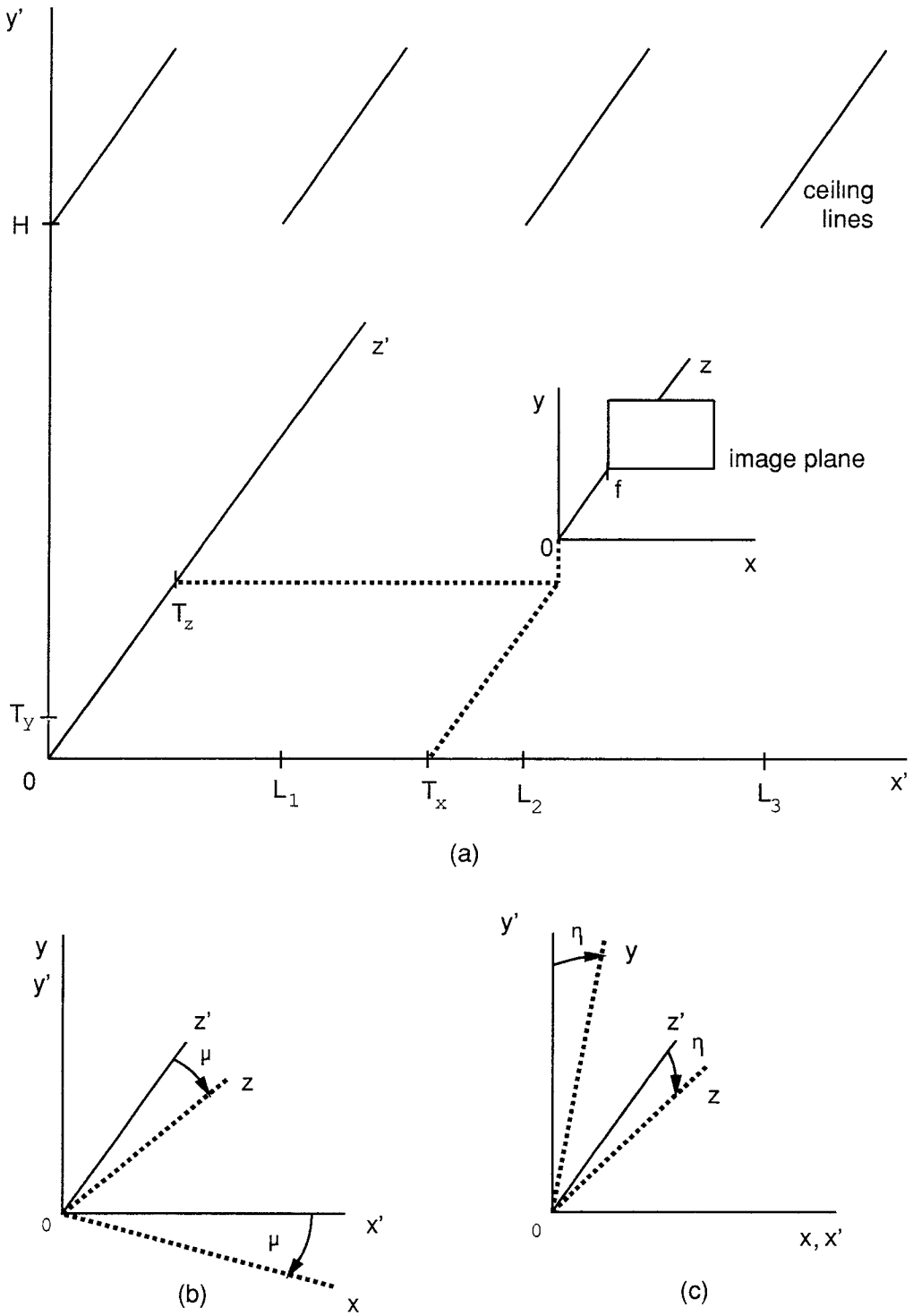

Figure 4.1: *Coordinate Systems: (a) Definition of the room coordinate system and camera coordinate system, (b)-(c) definition of the angles μ and η.*

we can transform P' into P by multiplying $\mathbf{p'_h}$ with the three matrices:

$$\mathbf{p_h} = \mathbf{p'_h}\ \mathbf{T}\ \Re_\mu\ \Re_\eta\ . \tag{4.5}$$

The matrix $\mathbf{T}$ essentially shifts the origin of the room system by $(T_x,\ T_y,\ T_z)$, such that it coincides with the origin of the camera system (Figure 4.1). The matrix $\Re_\mu$ represents the rotation around the y'-axis, where μ is the angle between the z-axis and the y', z'-plane. The matrix $\Re_\eta$ describes the rotation around the x'-axis, where η is the angle between the z-axis and the x', z'-plane. Note that the angle of each rotation is defined to be clockwise from the view-point of an observer who is looking along the positive x-or y-axis towards the origin.

Assuming that the camera is standing on the x', z'-plane (floor), we call μ the *turn* of the camera with respect to the y', z'-plane. Analogously, the angle between the camera and the x', z'-plane (floor) is called *tilt* (η). Hence, given the turn and the tilt of the camera, plus the translation $(T_x,\ T_y,\ T_z)$ of the origins, any point P' in the room system can be transformed into the corresponding point P in the camera system.

4.4 VANISHING POINTS WITH MOVING CAMERA

Vanishing points are detected in the camera system, based on parallel lines in the room system. In order to examine vanishing points when the camera is moving, the relationship between the lines in the camera system and the corresponding lines in the room system have to be defined. Equation 4.5 establishes the coordinate transform between points. This can easily be used to transform lines from the room system into the camera system . In general, a line, $\mathbf{l'} = \mathbf{l'_0} + \lambda'\ \mathbf{l'_d}$, in the

room system can be expressed by

$$\mathbf{l}' = \begin{bmatrix} x_0' \\ y_0' \\ z_0' \end{bmatrix} + \lambda' \begin{bmatrix} a' \\ b' \\ c' \end{bmatrix} . \tag{4.6}$$

The corresponding line, $\mathbf{l} = \mathbf{l_0} + \lambda\, \mathbf{l_d}$, in the camera system can be expressed by

$$\mathbf{l} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + \lambda \begin{bmatrix} a \\ b \\ c \end{bmatrix} . \tag{4.7}$$

Line $\mathbf{l}'$ in the room system can be transformed into line $\mathbf{l}$ in the camera system by the coordinate transform. To simplify the derivation, the vector $\mathbf{l_0'}$ to the base point of line $\mathbf{l}'$ is assumed to be $(L_n, H, 0)$. Without loss of generality, we can define equally use any vector pointing to any point on the line, l'. In addition, there are no assumptions made regarding the direction of the line $\mathbf{l}'$. The direction vector $\mathbf{l_d'}$ can point in any direction. Using Equation 4.5, we transform the vector $\mathbf{l_0'}$ (base point) and the direction vector $\mathbf{l_d'}$, separately. The vector $\mathbf{l_0'}$ (base point) can be transformed by

$$\mathbf{l_{0h}} = \mathbf{l_{0h}'}\, T\, \Re_\mu\, \Re_\eta\, , \tag{4.8}$$

where $\mathbf{l_{0h}'} = [L_n\ ,\ H\ ,\ 0\ ,\ 1]$ (the subscript h indicates homogeneous coordinates), and $\mathbf{T}$, $\Re_\eta$ and $\Re_\mu$ are defined in Equation 4.1, 4.2 and 4.3, respectively. Analogously, the direction vector $\mathbf{l_d'}$ (room system) can be transformed into $\mathbf{l_d}$ (camera system) using

$$\mathbf{l_{dh}} = \mathbf{l_{dh}'}\, \Re_\mu\, \Re_\eta\, , \tag{4.9}$$

where $\mathbf{l_{dh}'} = [a'\ ,\ b'\ ,\ c'\ ,\ 1]$. Here it is not necessary to apply the translation $\mathbf{T}$, since the direction vector is an independent vector, without a specific location associated with it. Equations 4.8 and 4.9 determine the parameters of line $\mathbf{l}$ in the camera system uniquely. Hence, after converting $\mathbf{l_{0h}}$, $\mathbf{l_{dh}}$ into Cartesian coordi-

nates, we can write the line equation of $\mathbf{l} = \mathbf{l_0} + \lambda\, \mathbf{l_d}$ as

$$\mathbf{l} = \begin{bmatrix} (L_n - T_x)\cos\mu + T_z \sin\mu \\ (H - T_y)\cos\eta + (L_n - T_x)\sin\mu\sin\eta - T_z\cos\mu\sin\eta \\ (T_y - H)\sin\eta + (L_n - T_x)\sin\mu\cos\eta - T_z\cos\mu\cos\eta \end{bmatrix} + \lambda \begin{bmatrix} a'\cos\mu - c'\sin\mu \\ b'\cos\eta + (a'\sin\mu + c'\cos\mu)\sin\eta \\ -b'\sin\eta + (a'\sin\mu + c'\cos\mu)\cos\eta \end{bmatrix}. \quad (4.10)$$

This is the transformation of line $\mathbf{l}'$ (room system) into line $\mathbf{l}$ (camera system). Assuming there is no translation and rotation ($T_x = T_y = T_z = \mu = \eta = 0$), it can be shown easily that the two lines $\mathbf{l}'$ (Equation 4.6) and $\mathbf{l}$ (Equation 4.10) are identical. Equation 4.10 describes the transformation of lines in terms of the location and orientation of the camera in the room system. This enables us to determine the location of the vanishing points as a function of the parameters of the line $\mathbf{l}$. We know from Chapter 3 that the vanishing point is defined as the line point with $\lambda \to \infty$, projected into the image ($z_\lambda = f$). Therefore, applying Equation 3.13 on line $\mathbf{l}$ given by Equation 4.10, the vanishing point in the image plane is

$$V_l = f \begin{bmatrix} \frac{a'\cos\mu - c'\sin\mu}{-b'\sin\eta + (a'\sin\mu + c'\cos\mu)\cos\eta} \\ \frac{b'\cos\eta + (a'\sin\mu + c'\cos\mu)\sin\eta}{-b'\sin\eta + (a'\sin\mu + c'\cos\mu)\cos\eta} \\ 1 \end{bmatrix}. \quad (4.11)$$

It is assumed that line $\mathbf{l}'$ is not parallel to the image plane which implies that the denominators in Equation 4.11 are not equal to zero [$-b'\sin\eta + (a'\sin\mu + c'\cos\mu)\cos\eta \neq 0$]. From Equation 4.11 it is clear that the vanishing point V_l is determined by the direction of the line l' [$l'_d = (a', b', c')$], and the turn and tilt of the camera in the room system (μ, η). The vanishing point V_l does not depend on the location of the line $\mathbf{l}'$ or the translation (T_x, T_y, T_z). Any set of parallel lines has the same vanishing point (see Subsection 3.2.2, Property 3). Therefore any line $\mathbf{l}'$ with the direction $\mathbf{l'_d}$ has the same vanishing point V_l no matter where it is located in the room. This is a very fundamental property of vanishing points

and has serious implications on the application of vanishing points in navigation.

It is clear that Equation 4.11 is valid for the very general case defined in the generic model (Section 4.2). There are no assumptions made concerning the orientation of the lines (a', b', c') and the orientation of the camera (μ , η). Hence, since only the minimum constraints of the generic model are made, this vanishing point can be called a *generalized vanishing point.*

4.4.1 Rotational Movement

To illustrate the properties of generalized vanishing points, Equation 4.11 is simplified by adding constraints to the generic model: We assume the lines are located in the ceiling of the room. Additional, the lines are assumed to be parallel to the z'-axis of the room system (Figure 4.1). The equation of the line $\mathbf{l}' = \mathbf{l}'_0 + \lambda' \mathbf{l}'_d$ can be rewritten as

$$\mathbf{l}' = \begin{bmatrix} L_n \\ H \\ 0 \end{bmatrix} + \lambda' \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} . \tag{4.12}$$

Using the simplified direction vector $\mathbf{l}'_d$ in Equation 4.11, the vanishing point V_l is defined by

$$v_i = f \begin{bmatrix} \frac{-\tan\mu}{\cos\eta} \\ \tan\eta \\ 1 \end{bmatrix} \tag{4.13}$$

Rewriting Equation 4.13, the components of the vanishing point V_l in the image plane are

$$x_i = \frac{-f\tan\mu}{\cos\eta} \tag{4.14}$$

and

$$y_i = f\tan\eta \tag{4.15}$$

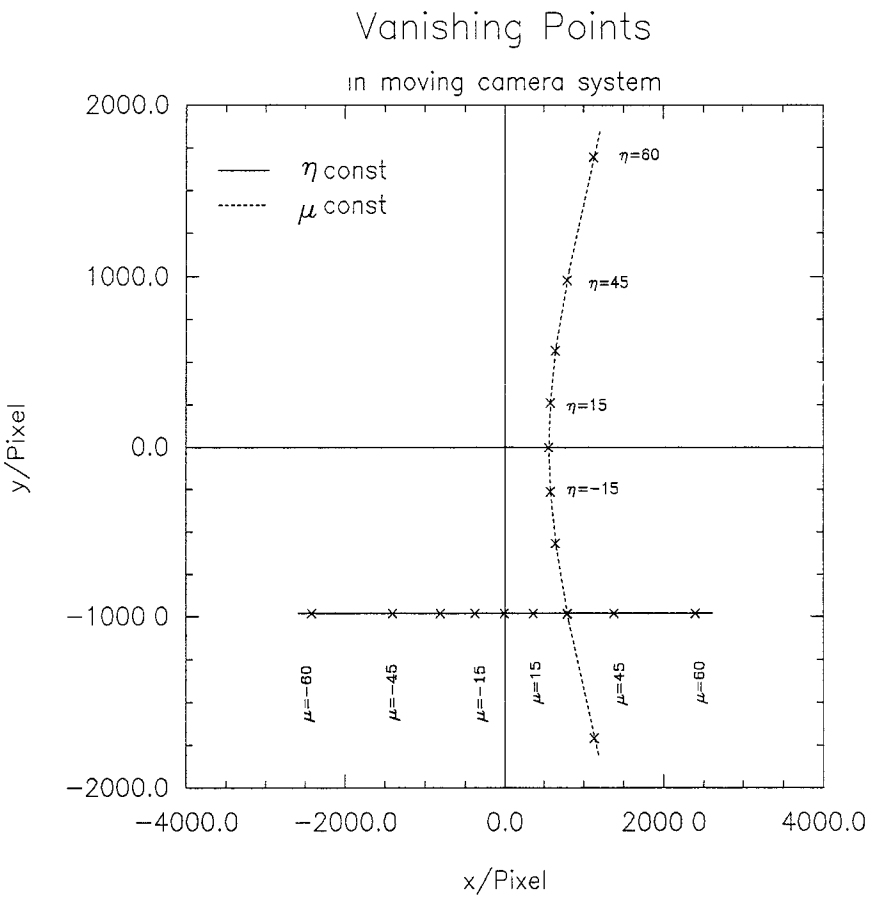

Figure 4.2: *Movement of vanishing points as a function of the angles μ and η.*

Now we can analyze the movement of vanishing points in the image plane for any given movement of the camera (μ , η). Assuming the camera is pointing in the direction of the z-axis ($\mu = \eta = 0$), the vanishing point is located at the origin of the image plane ($V_l = (0,0)$). If the camera is rotating (μ variable) with a constant tilt (η constant), the vanishing points form a horizontal line in the image plane (Figure 4.2(a)). For each rotation μ there exists a unique vanishing point on that horizontal line. If the camera tilt is varied (η variable) while there is no rotation (μ constant), the vanishing points are located on a symmetrically shaped curve in the image plane (Figure 4.2(a)). Figure 4.2(b) illustrates the movement of vanishing points for various (μ, η), with either one being held constant. If μ or η approach 90 degrees, the vanishing point approaches infinity. In any other cases, Equations 4.14 and 4.15 enable us to predict the location of the vanishing point for any given rotation μ and tilt η of the camera.

4.4.2 Translational Movement

It is an intrinsic property of vanishing points that they are insensitive to translation [CT90]. A translation of the camera in the room system does not change the location of the vanishing point. This results from the definition of the vanishing point. The vector $\mathbf{v_l}$ from the focal point toward the vanishing point in the image plane is parallel to the direction vector $\mathbf{l'_d}$ of the line in the room system (Equation 3.14). Since the direction vector $\mathbf{l'_d}$ of the line is an independent vector, the vanishing point has to be independent from the location of the camera and the location of the line. Therefore, any translation of the camera that does not change the direction of the optical axis (z-axis), has no influence on the vanishing point.

4.5 NAVIGATING THE ROBOT

The previous sections assume a rotating camera which is located on the floor (x, z-plane) of the room system. For robot navigation the camera is mounted on the robot such that it is pointing in the direction the robot moves to. The angle μ now represents the turn of the robot, the angle η still represents the tilt of the camera (Figure 4.1).

In this section we define a model of the robot environment which includes many constraints and assumptions. The next section generalizes the environment applying the generic model. The objective here is to navigate the robot parallel to the straight lines located on the ceiling of a room. For simplicity, let the lines be parallel to z-axis of the room (Figure 4.1). These lines can be some kind of a ceiling pattern or a ceiling grid. The camera is pointing up to the ceiling ($\eta < 90°$) and takes pictures of the ceiling lines. Line segments are extracted by the edge detection and line fitting algorithms (Chapter 2). Intersections between pairs of line segments are then determined by the modified cross-product method (Section 3.5) . This algorithm provides the location of the vanishing point of the ceiling lines. In the previous section, we discussed how to compute the vanishing point for a given turn and tilt. Here, the vanishing point is given, and the turn and tilt of the robot are to be computed. This can be done by rewriting Equations 4.14 and 4.15 as

$$\mu = \arctan\left(\frac{-x_i \cos\eta}{f}\right) \tag{4.16}$$

and

$$\eta = \arctan\left(\frac{y_i}{f}\right) . \tag{4.17}$$

However, the cross-product method does not directly provide the (x_i, y_i) location of the vanishing point. Instead, the vanishing point in azimuth and elevation rep-

resentation (α, β) on the Gaussian sphere is given. To convert the angles (α, β) into a (x_i, y_i) coordinate in the image plane, we can use the following relationship

$$x_i = f \frac{\cos \alpha}{\tan \beta}, \tag{4.18}$$

$$y_i = f \frac{\sin \alpha}{\tan \beta}. \tag{4.19}$$

Substituting Equation 4.18 and 4.19 into Equations 4.16 and 4.17 yields

$$\mu = \arctan \left(\frac{\cos \alpha \ \cos \eta}{\tan \beta} \right), \tag{4.20}$$

$$\eta = \arctan \left(\frac{\sin \alpha}{\tan \beta} \right). \tag{4.21}$$

In order to compute the turn μ, we need the location (α, β) of the vanishing point and the tilt of the camera (Equation 4.20). It is obvious from Equation 4.14 and Figure 4.1 that μ is a function of the location of the vanishing point and the tilt of the camera. In Equation 4.21 we only need the location of the vanishing point (α, β) to determine the tilt η. For the navigation of the robot the tilt of the camera is assumed to be constant. Equation 4.20 can thus be used to compute the rotation of the robot necessary to align it with the lines. Figure 4.3(a) shows two ceiling lines and the robot in two positions. In Position 0, it is facing the desired direction, parallel to the ceiling lines ($\mu_0 = 0°$). Therefore, the vanishing point VP_0 is located on the y-axis of the image plane (Figure 4.3(b)). Because of the tilt of the camera ($\eta_0 = -35°$), the vanishing point is located below the origin. In Position 1 the robot is turned to the left ($\mu_0 = 30°$). This results in a vanishing point shifted to the right. Figure 4.3(c) illustrates that movement of the vanishing point (VP_1). Based on the vanishing point (VP_1) and the tilt η, Equation 4.20 can be used to determine the angle μ_v between the ceiling lines and the direction the robot is directed to. Since the robot should be navigated parallel to the lines, it

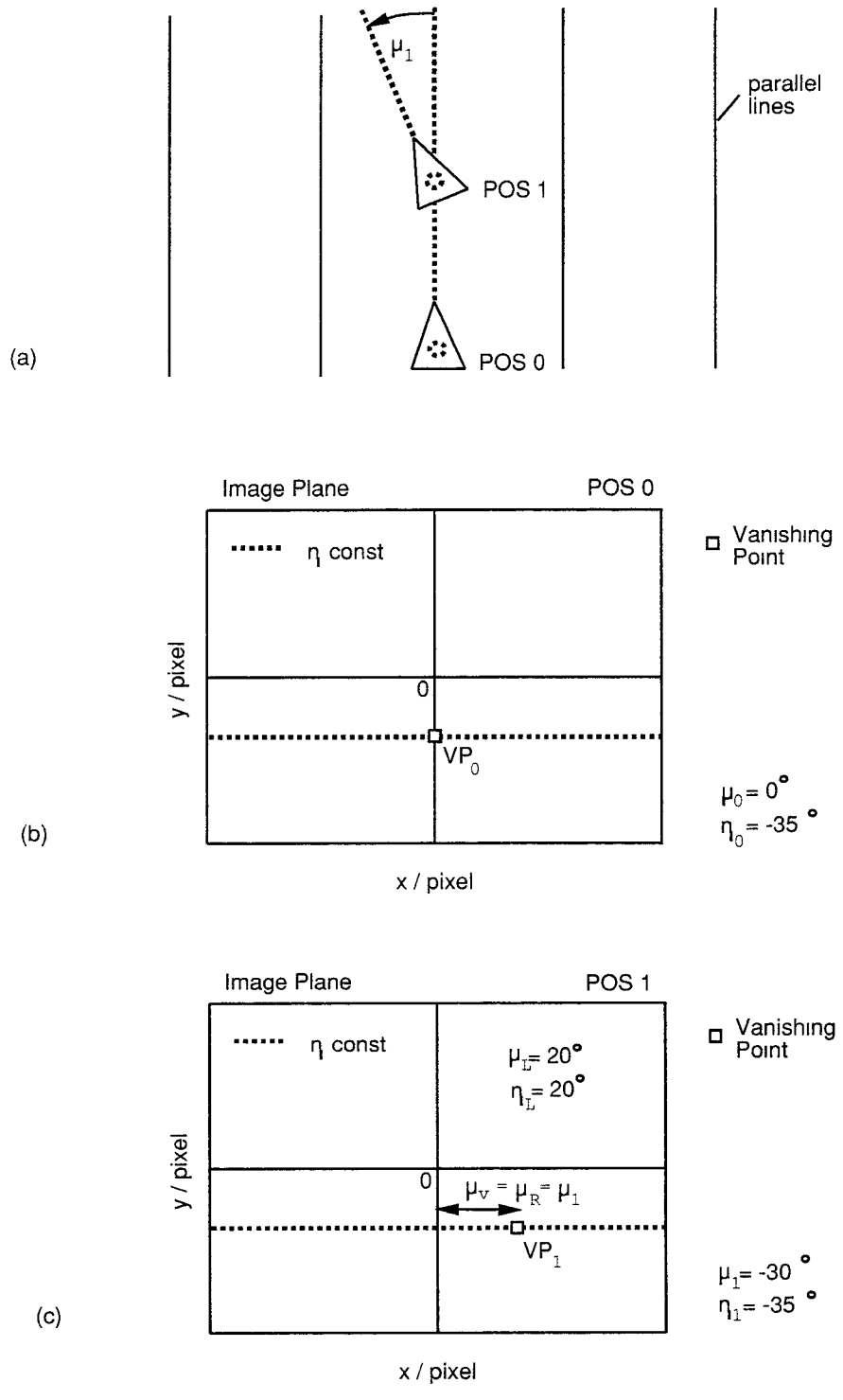

Figure 4.3: *Vanishing points used for robot navigation: (a) robot in two positions, (b)-(c) vanishing point in the image plane for robot position* 1 *and* 2.

has to be rotated back into the direction of the lines with

$$\mu_R = \mu_v \,, \tag{4.22}$$

where μ_R denotes the angle of the robot rotation. Equation 4.22 is called the *navigation criterion*. This method of computing the rotation μ_n can be used in a recursive navigation procedure.

The procedure starts with taking a picture from the ceiling, finding the vanishing points and computing the robot rotation μ_R. In the next step the robot rotates μ_R degrees which aligns the robot with the ceiling lines (see Figure 4.3 for illustration), and then it moves forward. Then, a picture is taken from this new position of the robot, and μ_R is computed again. The robot rotates and moves forward and again the procedure repeats itself such that the vanishing points are used as a guiding reference for the direction of the robot.

4.6 GENERALIZATION OF THE NAVIGATION ENVIRONMENT

The generalization described in this section is twofold. First, the direction that the robot should move toward is not restricted to be parallel to the lines. Any direction can be chosen except the direction perpendicular to the lines (vanishing points at infinity). Second, the robot environment is generalized according to the generic model (Section 4.2). We assume an environment with a minimum number of constraints. This ensures that the navigation method is appropriate for a large class of different environments and applications.

The generic model requires that a minimum of two straight parallel lines are to be detected by the camera. In addition, the direction of the lines are known with

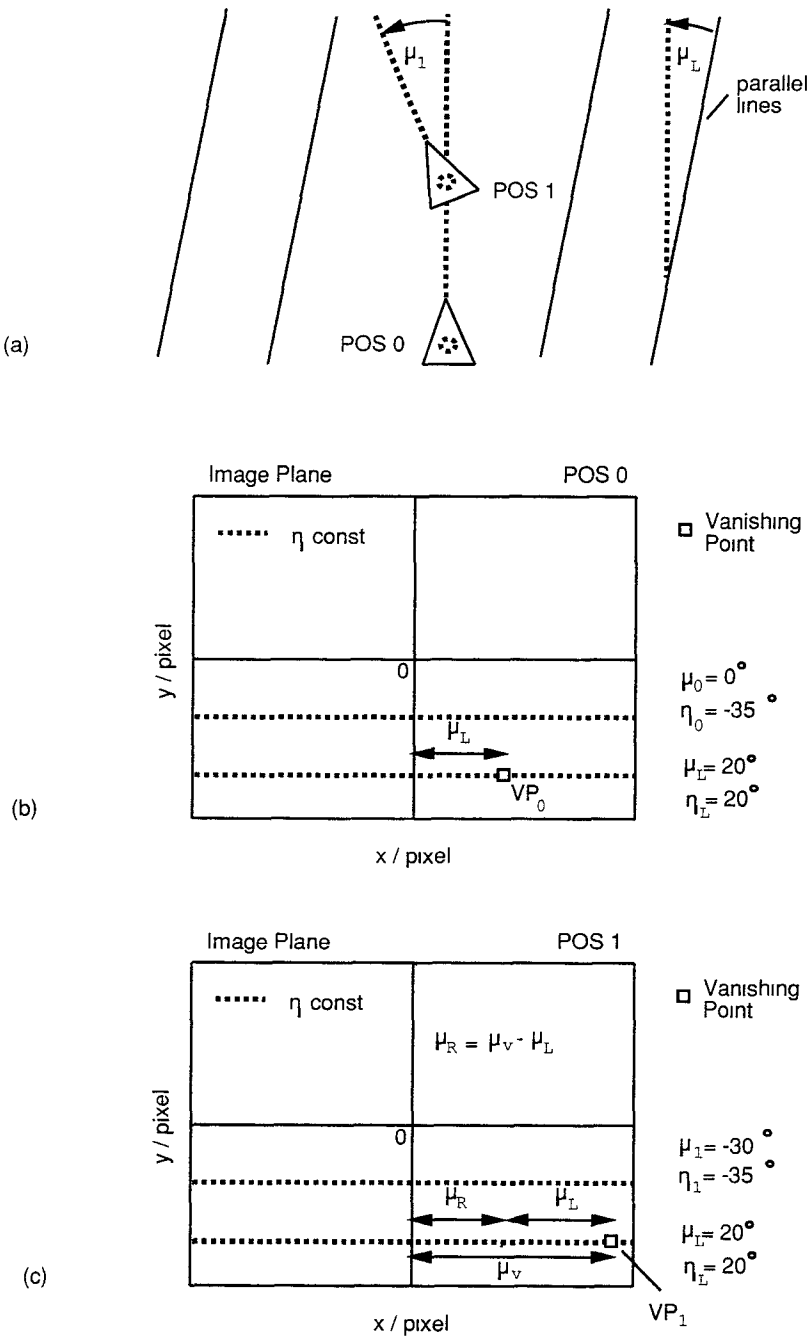

Figure 4.4: *Vanishing points used for robot navigation with rotated and tilted parallel lines* (μ_L, η_L). *(a) robot in two positions, (b)-(c) vanishing point on image plane for robot position* 1 *and* 2.

respect to the robot environment. The direction of the lines is represented by two angles μ_L and η_L, where μ_L denotes the angle with respect to the y', z'-plane and η_L represents the tilt with respect to the x', z'-plane. Again, it has to be ensured that the lines are not parallel to the image plane of the camera. With this minimum number of constraints assumed on the environment of the robot, it is possible to redefine the navigation criterion in Section 4.5.

Similar to Section 4.5, the vanishing point is given in azimuth and elevation representation (α , β). In addition, the direction of the lines in the room is known, and denoted as μ_L and η_L (see Figure 4.4). Now it is possible to analyze the movement of the vanishing points considering both the orientation of the camera (μ_n , η_n) and the orientation of the lines (μ_L , η_L). The location of the vanishing point is based on the direction of the lines with respect to the camera system. This implies that a rotation of the camera system by μ degrees results in the same vanishing point as a rotation of the line by $-\mu$ degrees. For example, a rotation of the camera system by $\mu_1 = 30°$ results in the vanishing point (α_1, β_1). If we rotate the direction of the lines (instead of the camera) by $\mu_L = -30°$, the vanishing point is in the same location (α_1, β_1). In order to find the orientation of the robot with respect to the direction of the lines, we have to subtract the angle μ_n by μ_L as follows

$$\mu_v = \mu_n - \mu_L , \tag{4.23}$$

where μ_V includes both the rotation of the camera and the rotation of the lines. Similarly for the tilt of camera and the tilt of the lines, we can use

$$\eta_v = \eta_n - \eta_L , \tag{4.24}$$

to compute the tilt η_v. Note that the angles μ_v and η_v represent the turn and tilt of the camera relative to the lines, and therefore can be used to determine the location

of the vanishing point using Equations 4.14 and 4.15.

Conversely, given a vanishing point we can use Equation 4.21 to compute η_v, and then Equation 4.20 to find μ_v. Now it is possible to rewrite the Equations 4.23 and 4.24 to determine μ_n and η_n (with μ_L and η_L known). For navigation purposes we are interested in the rotation μ_R necessary to rotate the robot such that it points in the desired direction (parallel to the z-axis). Since we rotate the robot toward the z-axis, the angle of rotation is equivalent to $-\mu_n$. Using the above findings and Equation 4.23, we can compute the angle of rotation by

$$\mu_R = \mu_v - \mu_L . \tag{4.25}$$

This equation is called the generalized navigation criterion. Figure 4.4 illustrates this concept. Comparing Figure 4.3(b) and Figure 4.4(b), both diagrams show the vanishing point for the robot pointing in the direction of the z-axis (POS 0). The vanishing points are not in the same location because the lines in Figure 4.3(a) are turned and tilted ($\mu_L = \eta_L = 20°$). Note that the vanishing point is shifted down and right.

In Position 1 (Figure 4.4(a),(c)), the robot is turned left ($\mu_1 = -30°$). This moves the vanishing point VP_1 further to the right. As described above, the vanishing point VP_1 given in (α , β) representation, can be converted into μ_v using Equation 4.20. Note that the tilt η consists of the tilt of the camera minus the tilt of the lines ($\eta = \eta_n - \eta_L$). Assuming that the robot has to be turned back in the direction it had in Position 0, we can calculate the rotation μ_R using Equation 4.25.

The generalized navigation criterion is valid for a large class of environments defined in the generic model. It also permits to chose any direction for the robot to move in (except perpendicular to the lines).

4.7 RESULTS AND CONCLUSION

The robot used for the navigation experiments is HERO 2000 (Model ET-19, manufactured by Heath Company). The navigation process is implemented on a PC, with the software written in C. The robot and the camera are linked to the PC via cable connections. The navigation process assumes an environment with the lines located in the ceiling of an indoor scene, where the robot is supposed to move in a direction parallel to the lines (Figure 4.1). Figure 4.5 shows images of a ceiling pattern with the robot pointing in three different directions. The line segments have been detected with the edge detection and line fitting algorithms described in Chapter 2. Refer to Figure 3.4 for an original image of the ceiling pattern. In the top image of Figure 4.5 the robot is turned to the left ($\mu_0 = -30°$). In the middle image, the robot is facing the direction parallel to the lines ($\mu_0 = 0°$). In the bottom image, the robot is turned to the right ($\mu_0 = 30°$). Figure 4.6 shows the results of the detected vanishing points corresponding to the three respective directions which the robot points to (images in Figure 4.5). The accepted intersections are displayed in the accumulator array representing azimuth and elevation (α, β) on the Gaussian sphere (see Section 3.5 for explanation). The vanishing point is selected with the window shown in the accumulator. The location of the vanishing point within the window is marked with a cross. From Figure 4.6, it is obvious that a robot turning to the left (negative μ) results in the vanishing point being shifted to the right. Conversely a robot turn to the right (positive μ) results in shifting the vanishing point to the left. The (α, β) location of the vanishing point is used to compute the angle between the lines and the direction the robot is pointing to (Equation 4.20). The experimental results for computing the rotation μ_R from detected vanishing points are given in Figure 4.7. The graph shows

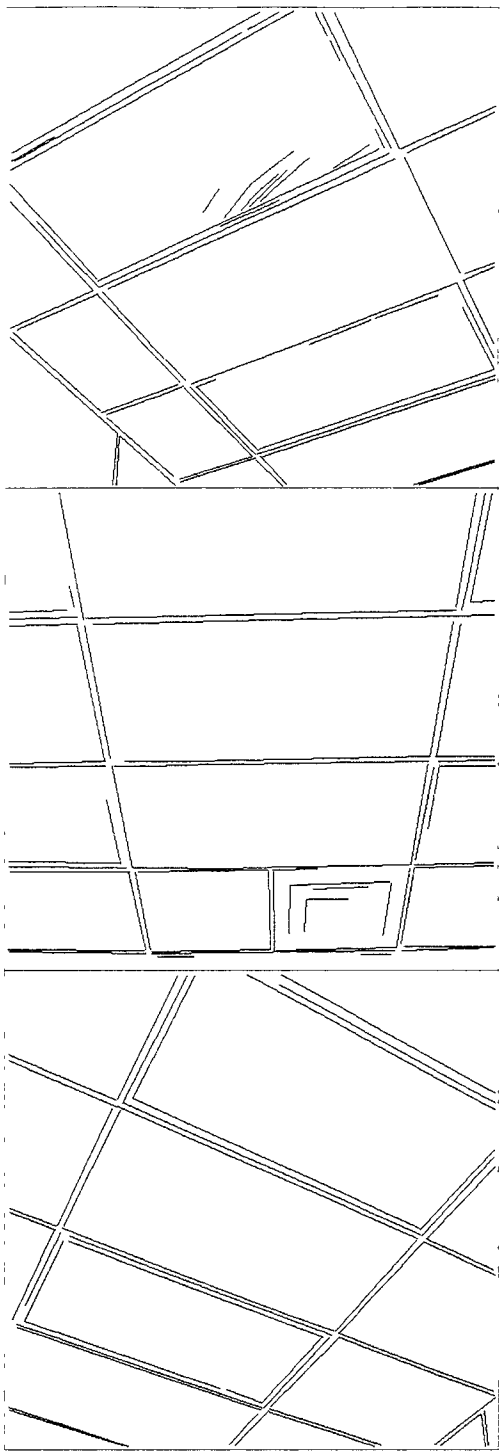

Figure 4.5: *Detected lines in ceiling pattern with the robot facing three different directions, $\mu = -30°$ (top), $\mu = 0°$ (middle) and $\mu = +30°$ (bottom)*

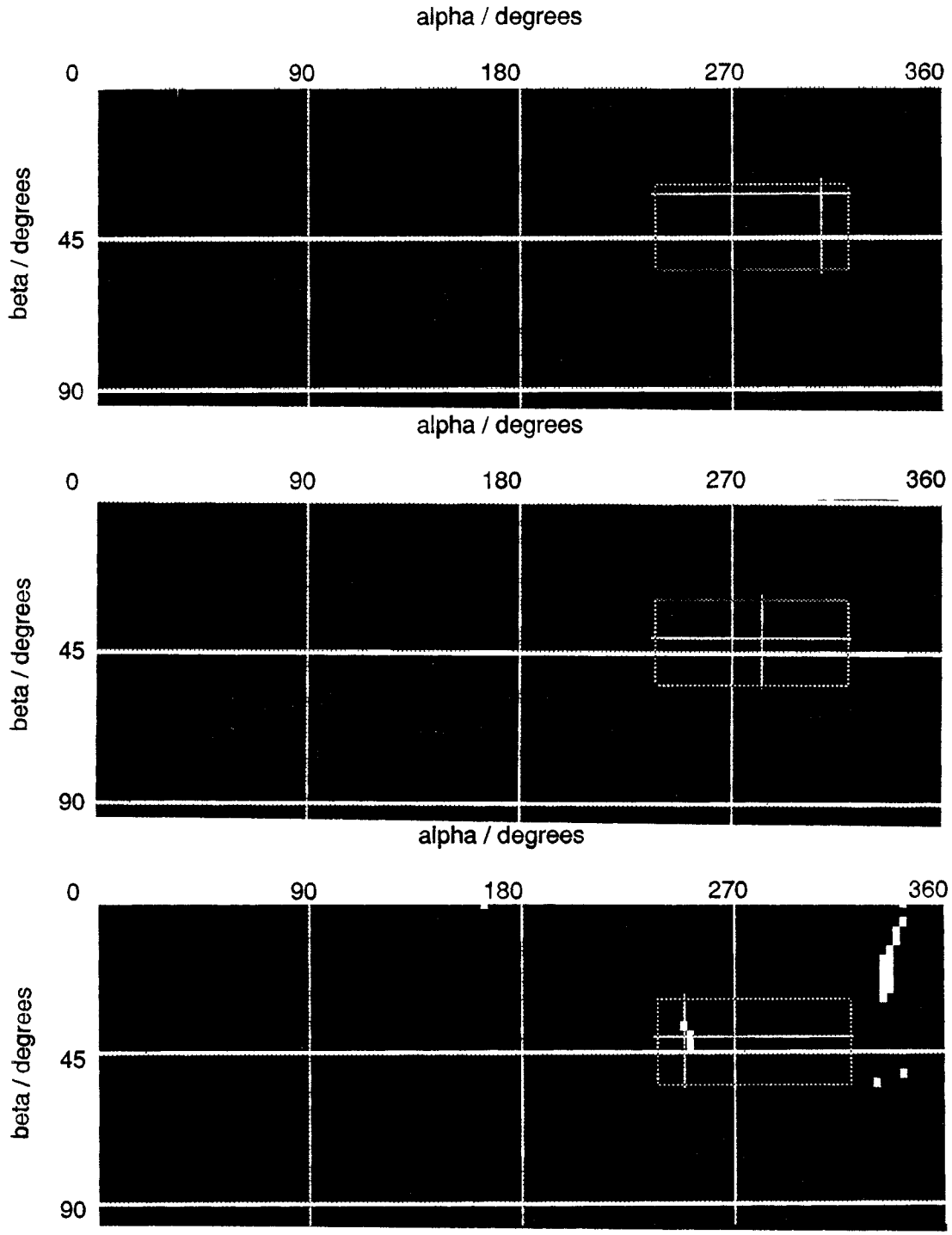

Figure 4.6: *Detected intersection and vanishing points with the robot facing three different directions, $\mu = -30°$ (top), $\mu = 0°$ (middle) and $\mu = +30°$ (bottom)*

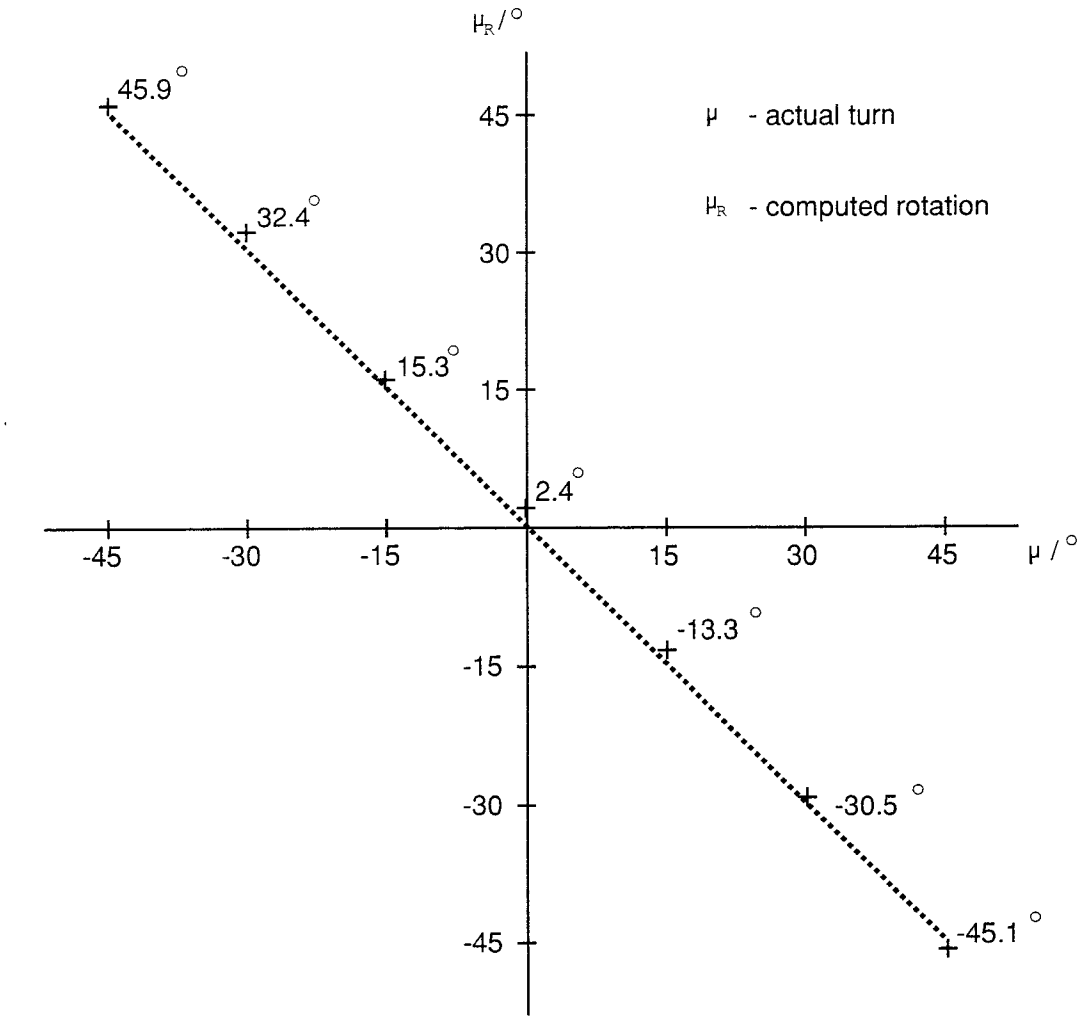

Figure 4.7: *Experimental results of using vanishing points to determine rotation μ_R for robot turn μ (see text).*

the computed μ_R as a function of robot turns μ, where μ_R represents the rotation necessary to point the robot into the direction of the lines. The error associated with the computed μ_R is approximately ± 2.5 degrees. This is sufficient for navigating the robot considering the inaccuracies of the robot movements itself (robot translation and rotation are inaccurate). These results demonstrate the usefulness of vanishing points in robot navigation. Vanishing points provide a reference for detecting robot rotation. This is true not only for parallel ceiling lines, but in general for any two parallel lines in indoor or outdoor environments. This opens a vast

field of applications of this navigation technique. Like any other sensor, vanishing points have intrinsic limitations in their sensing capabilities. They are not sensitive to translation which imposes a serious problem in navigation. The robot can be shifted towards a wall or an obstacle, and the vanishing point would still be in the same place. Hence, integrating the vanishing point techniques with other sensors are recommended.

Bibliography

[Bad74] N. Badler. Three-dimensional motion from two-dimensional picture sequences. Technical report, Department of Computer Science, University of Toronto, 1974.

[Bar83] S. T. Barnard. Methods for interpreting perspective images. *Journal of Artificial Intelligence (Netherlands)*, 21(4):435–462, November 1983. ISSN: 0004-3702.

[Bre84] V. Brezins. Accuracy of laplacian edge operators. *Computer Vision, Graphics and Image Processing*, 27:195–210, August 1984.

[Bro89] R. A. Brooks. Engineering approach to building complete, intelligent beings. In *Proc. SPIE - Int. Soc. Opt. Eng. (USA)*, pages vol.1002, 618–625, 1989.

[CT90] B. Caprile and V. Torre. Using vanishing points for camera calibration. *International Journal of Computer Vision*, pages 127–139, 1990.

[Elf87] A. Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics & Automation (USA)*, RA-3(3):249–265, June 1987. 0882-4967/87/0600-0249$01.00.

[GW87] R. C. Gonzalez and P. Wintz. *Digital Image Processing.* Addison Wesley Publishing Company, second edition, 1987.

[HL86] R. Hummel and D. Lowe. Computing gaussian blur. In *Proceedings of the Eighth International Conference on Pattern Recognition (Cat. No 86CH2342-4)*, pages 910–912. IEEE Computer Society, IEEE Computer Society Press, October 1986. ISBN: 0 8186 0742 4.

[HM86] J. S. Chen; A. Hertas and G. Medioni. Very fast convolution with laplacian-of-gaussian masks. In *Proceedings CVPR '86: IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No.86CH2290-5)*, pages 293–298. IEEE Computer Society, IEEE Computer Society Press, June 1986. ISBN 0 8186 0721 1.

[KB88] D. J. Kriegman and T. O. Binford. Generic models for robot navigation. In *Proceedings of the 1988 IEEE International Conference on Robotics and Automation (Cat. No.88CH2555-1)*, pages Vol.2, 746–751. Robotics Laboratory, Department of Computer Science, Stanford University, IEEE Comput. Soc. Press, Washington, DC, USA, 1988. CH2555-1/88/0000-0746$01.00; ISBN 0 8186 0852 8.

[Ken79] J. R. Kender. Shape from texture: An aggregation transform that maps a class of textures into surface orientation. Technical report, Department of Computer Science, Carnegie-Mellon University, 1979.

[KK82] S. A. Shafer; T. Kanade and J. R. Kender. Gradient space under orthographie and perspective. In *Proceedings of the Workshop on Computer Vision: Representation and Control*, pages 26–34. IEEE, New York, NY, USA, 1982. CH1793-9/82/0000/0026$00.75.

[MA84] M. J. Magee and J. K. Aggrawal. Determining vanishing points from perspective images. *Computer Vision, Graphics, and Image Processing*, vol.26, no.2:256–67, 1984.

[ME89] L. Mattheis and A. Elfes. Probabilistic estimation mechanisms and tesselated representations for sensor fusion. In *Proc. SPIE - Int. Soc. Opt. Eng. (USA)*, pages Vol.1003, 2–11, 1989.

[NR88] R. S. Weiss; H. Nakatani and E. M. Riseman. An error analysis for surface orientation from vanishing points. In *Proc. SPIE Int. Soc. Opt. Eng. (USA) vol.974*, pages 187–194, 1988.

[QM89] L. Quan and R. Mohr. Determining perspective structures using hierarchical hough transform. *Pattern Recognition Letters (Netherlands)*, 9(4):279–286, May 1989. 0167-8655/89/$3.50.

[Rei91] Harald Reinhard. Digitale bildverarbeitung, kamera kalibrierung kamera-kalibrierung für ein stereo-vision system. Master's thesis, Fachhochschule Würzburg-Schweinfurt, March 1991.

[RK82a] A. Rosenfeld and A. C. Kak. *Digital Picture Processing Volume 1.* Academic Press, Inc., second edition, 1982.

[RK82b] A. Rosenfeld and A. C. Kak. *Digital Picture Processing Volume 2.* Academic Press, Inc., second edition, 1982.

[SK80] H. Nakatani; S. Kimura; O. Saito and T. Kitahashi. Extraction of vanishing point and its application to scene analysis based on image sequence. In *Proceedings of the 5th international conference on pattern recognition*, pages 370–372. Department of Information Science, Shizuoka University, IEEE, New York, USA, 1980.

[SS88] A. Singh and M. Shneier. A linear feature extraction scheme for the PIPE. Technical report, Robotics and Flexible Automation Department, Philips Laboratories, March 1988.

[WH88] G. Wei and Z. He. Determining vanishing point and camera parameter: New approaches. In *9th International Conference on Pattern Recognition (IEEE Cat. No.88CH2614-6)*, pages 450–452. Int. Assoc. Pattern Recogition, IEEE Comput. Soc. Press, Washington, DC, USA, 1988. CH2614-6/88/0000-0450$01.00; ISBN 0 8186 0878 1.

[WR84] H. Nakatani; R. S. Weiss and E. M. Riseman. Application of vanishing points to 3-d measurement. In *Proc. SPIE Int. Soc. Opt. Eng. (USA)*, pages Vol.507, 164–169, 1984.