

1991

Low to medium level image processing for a mobile robot

Cecilia H. Espinosa
Portland State University

Let us know how access to this document benefits you.

Follow this and additional works at: https://pdxscholar.library.pdx.edu/open_access_etds



Part of the [Electrical and Computer Engineering Commons](#)

Recommended Citation

Espinosa, Cecilia H., "Low to medium level image processing for a mobile robot" (1991). *Dissertations and Theses*. Paper 4294.

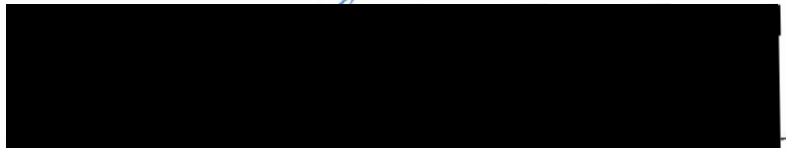
[10.15760/etd.6178](https://pdxscholar.library.pdx.edu/etd/10.15760/etd.6178)

This Thesis is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. For more information, please contact pdxscholar@pdx.edu.

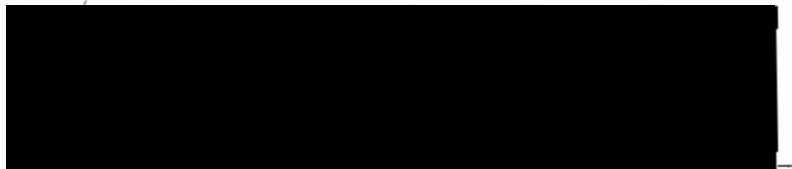
AN ABSTRACT OF THE THESIS OF Cecilia H. Espinosa for the Master of Science in
Electrical and Computer Engineering presented May 22,1991.

Title: Low To Medium Level Image Processing for a Mobile Robot.

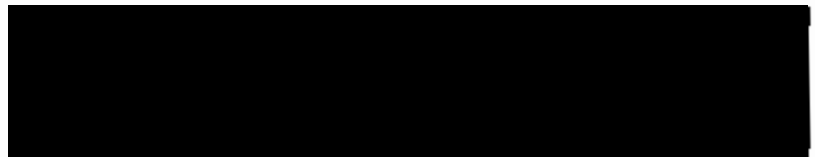
APPROVED BY THE MEMBERS OF THE THESIS COMMITTEE:



Marek A. Perkowski, Chair



Michael A. Driscoll



Jean Scholtz

The use of visual perception in autonomous mobile systems was approached with caution by mobile robot developers because of the high computational cost and huge memory requirements of most image processing operations. When used, the image processing is implemented on multiprocessors or complex and expensive systems, thereby requiring the robot to be wired or radio controlled from the computer system base.

Hereby developed is a simple, inexpensive and automatic image processing subsystem that is completely implementable on a PC-386SX with 640K base memory. The subsystem will serve as the front end of the vision system for the PSUBOT, an autonomous wheelchair robot currently being developed in the Department of Electrical Engineering, Portland State University. The low to medium level image processing of indoor scene images yields a compact scene description that will be used by the scene analysis and recognition subsystem to allow the robot to initially "learn" its environment and later recognize or confirm its bearings in the known environment.

The software component of the image processing subsystem is a systematic sequencing of some image preprocessing programs that were developed by past students and parts of Benjamin Dawson's SIMPP2 package for interfacing with the vision hardware. An automatic binarization thresholding program was written to completely automatize the preprocessing sequence and a line feature extraction scheme was developed and implemented to extract and characterize the line features that comprise the scene description. The hierarchical approach to line feature extraction was based on the fundamental concepts of the Hough Transform method and pyramids. The scheme proved to be efficient in extracting the salient line features in the scene and proved robust even in images with relatively poor quality. The format by which the results are passed to the scene recognition subsystem preserves the hierarchical classification of the line features, thereby allowing the subsystem to exploit that aspect in the course of the recognition task.

LOW TO MEDIUM LEVEL IMAGE PROCESSING

FOR A MOBILE ROBOT

by

CECILIA H. ESPINOSA

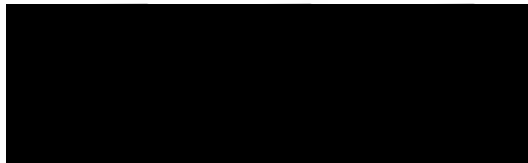
A thesis submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE
in
ELECTRICAL AND COMPUTER ENGINEERING

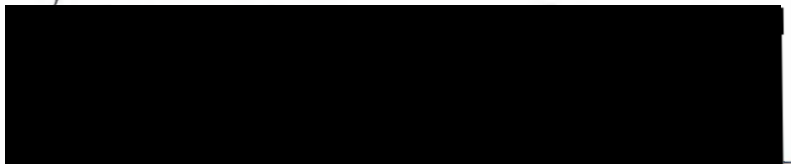
Portland State University
1991

TO THE OFFICE OF GRADUATE STUDIES:

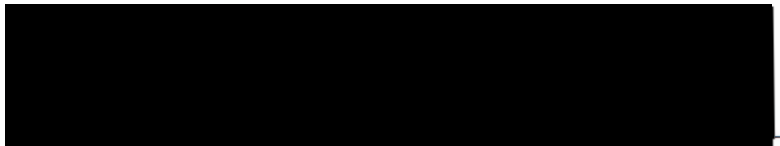
The members of the Committee approve the thesis of Cecilia H. Espinosa
presented May 22, 1991.



Marek A. Perkowski, Chair

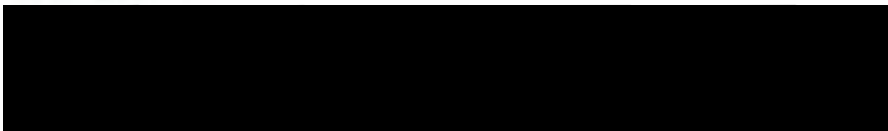


Michael A. Driscoll

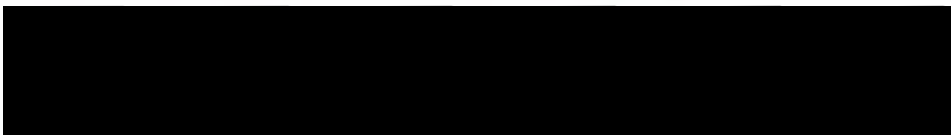


Jean Scholtz

APPROVED:



Rolf Schaumann, Chair, Department of Electrical Engineering



C. William Savery, Vice Provost for Graduate Studies and Research

ACKNOWLEDGEMENTS

I wish to express my profound gratitude to my adviser, Dr. Marek A. Perkowski, who patiently guided me through this endeavor, allowing me to be ambitious, creative and realistic, and whose unwavering support served as my beacon especially in the most trying times.

I also wish to thank Dr. Michael Driscoll and Dr. Jean Scholtz for their comments, suggestions and critiques which bolstered my spirit and inspired me to continue working on this subject even after the completion of this thesis.

In addition, I express my appreciation to the students working on the PSUBOT project especially Diudonne Mayi and Kevin Stanton for their ideas, suggestions and invaluable support; the past EE students: Kelly Spiller, Alvin Legate and Shiliang Wang whose experiences with their image processing projects helped me a lot; and the staff of the Department of Electrical Engineering who in one way or another provided their support towards the completion of this thesis.

I especially thank the U.S. Agency for International Development, whose sponsorship made possible my completion of this degree and The ASIA Foundation who patiently managed my training program.

Finally, I especially thank Mohammed Ishaque for the invaluable support and assistance he provided in the course of the preparation of this thesis.

Portland, Oregon

Cecilia H. Espinosa

TABLE OF CONTENTS

	PAGE
ACKNOWLEDGEMENTS	iii
LIST OF TABLES	vii
LIST OF FIGURES	viii
 CHAPTER	
I INTRODUCTION	1
Mobile Robots	2
PSUBOT: The Autonomous Wheelchair Robot	3
Problem Statement	4
Goals and Design Development Methodology	5
Evaluation Criteria.....	7
II THE IMAGE PROCESSING SYSTEM	9
Hardware Aspects	9
Software Aspects	10
An Overview of the Image Processing System	12
III IMAGE ENHANCEMENT	17
Histogram Equalization	18
Image Smoothing	21
IV EDGE DETECTION AND EXTRACTION	28
Edge Detection	28
Binarization	33

	Automatic Threshold Selection	34
	Edge Thinning	43
	Labelling	46
	The Edge Detection and Extraction Sequence	47
V	FEATURE EXTRACTION AND CHARACTERIZATION	48
	Review of Line Feature Extraction Techniques	48
	The Hough Transform Method	50
	Approaches to the Use of the Hough Transform Method	53
	The Line Extraction Scheme	55
	The Implementation of the Hierarchical Hough Transform Scheme	64
	Impact of the Initial Subimage Size on the Line Feature Extraction	75
	Feature Characterization	79
VI	EVALUATION OF RESULTS	81
VII	CONCLUSIONS AND FUTURE WORK	90
	Conclusions	90
	Future Work	92
	REFERENCES	94
	APPENDICES	
A	SET-UP FOR IMAGE ACQUISITION	98
B	THE PROGRAM SOURCE CODE FOR THE BINARIZATION THRESHOLD DETERMINATION USING THE BETWEEN-CLASS-VARIANCE METHOD	101
C	THE PROGRAM SOURCE CODE FOR THE BINARIZATION THRESHOLD DETERMINATION USING THE ENTROPY-BASED METHOD	104

D	DATA STRUCTURE DEFINITIONS FOR THE HIERARCHICAL REPRESENTATION OF LINE FEATURES	107
E	THE MAIN PROGRAM SOURCE CODE FOR THE HIERARCHICAL LINE EXTRACTION SCHEME USING THE HOUGH TRANSFORM METHOD (HHOUGH)	110
F	THE PROGRAM SOURCE CODE FOR THE LOWEST LEVEL LINE SEGMENT DETERMINATION ROUTINE OF HHOUGH	116
G	THE PROGRAM SOURCE CODE FOR THE HIGHER LEVEL LINE SEGMENT GROUPING ROUTINE OF HHOUGH	125
H	THE PROGRAM SOURCE CODE FOR THE PRINTING AND REMAPPING ROUTINES FOR LINES EXTRACTED BY HHOUGH	141
I	THE PROGRAM SOURCE CODE FOR THE POSTPROCESSING ROUTINES FOR THE HIERARCHY OF LINES GENERATED BY HHOUGH	149
J	A SAMPLE OUTPUT OF THE POSTPROCESSED HIERARCHY OF LINES	162

LIST OF TABLES

TABLE		PAGE
I	Binarization Threshold Determination Results	41
II	Settings for the Hough Transform Quantized Parameter Space	69
III	Processing Times for the Line Feature Extraction at Different Initial Subimage Sizes	75
IV	Number of Lines Extracted by the Hierarchical Scheme at Different Initial Subimage Sizes	76
V	Execution Times for the Image Processing Operations on Image1	81
VI	Effect of the Binarization Threshold [T], Smoothing and Labelling on the Thinning Time [s] and Number of Passes [P] for Image2	86
VII	Effects of Smoothing and Labelling on the Processing Times of the Line Feature Extraction Routines for Image1	87
VIII	Effects of Smoothing and Labelling on the Number of Lines Extracted from Image1	88
IX	Effects of Labelling on the Execution Times for the Image Processing Operations	89

LIST OF FIGURES

FIGURE		PAGE
1.	Functional Components of the PSUBOT	4
2.	Hardware Components of the Image Processing System	10
3.	The Vision System for the PSUBOT	12
4.	The Low to Medium Level Image Processing Steps	14
5.	The Mapping of a Gray level Histogram H to a Uniform Histogram G Using the Histogram Equalization Technique.....	20
6.	A Test Image	22
7.	The Test Image After Histogram Equalization	23
8.	The Gray Level Histograms of the Images in Figures 6 and 7	24
9.	Median Filter Geometries Used by Bovik, Huang and Munson	25
10.	Effect of Median Filtering on the Image in Figure 7	27
11.	The 2×2 Convolution Masks for the Digital Derivative Approximation	30
12.	Differential Edge Operators	31
13.	Mapping of the 3×3 Pixel Neighborhood	32
14.	The Gradient Image of Figure 10(a)	33
15.	The Gradient Image to be Binarized	41
16.	The Binarized Images	42
17.	Positional Mapping of the 3×3 Neighborhood of a Contour Point P1	44
18.	Thinning Effect on an Image.....	45
19.	The Effect of the Labelling Routine	47

20.	Straight Line Parameterizations	52
21.	Grouping of 2×2 Subimage Neighborhoods into Bigger Subimages	56
22.	Neighborhood of Subimages Used to Form Line Segment Groups for a Parent Subimage	57
23.	Overlapping Regions for Adjacent Central Subimages	57
24.	Hierarchical Representation of the Hierarchical Structure	58
25.	An Overlapped Subimage for the Lowest Level Line Segment Determination	60
26.	A Neighborhood of Subimages Containing Collinear Line Segments	62
27.	A Binary Line-thinned Image	72
28.	Remaps of the Line Segments Detected at Levels 1, 2, 3, and 4 for Image in Figure 27	73
29.	Remaps of the Line Segments Detected at Levels 5, 6, 7 and All the Levels for Image in Figure 27	74
30.	Binary Line-thinned Images Used for Initial Subimage Size Analysis	76
31.	The Level 1 Line Segments Detected with Varied Initial Subimage Sizes	77
32.	Detectability of the Lines at Higher Levels	78
33.	Image1	82
34.	Image2	85
35.	The Effect of Smoothing on Binarized Images	86

CHAPTER I

INTRODUCTION

The field of digital image processing has grown tremendously in the past decades, stirred by the increased capabilities computer technology afforded. Image processing enthusiasts have swamped the field with ideas, techniques and algorithms geared towards such aims as the improvement of the pictorial information for human interpretation [1], or processing of scene data for autonomous machine perception. Its diverse applications in the pure and applied sciences, medical diagnosis, military activities, robotics, rehabilitation, etc. [2], have led to more developments in the field, most of which are rather application oriented.

Robotics itself developed independently, initially concerned with mechanical effectors and manipulators for the machine assembly and eventually extended to autonomous mobile systems [3, 4]. Artificial intelligence allowed the use of image processing to endow the robots with "sight" and made it possible for the robot to react to the "visual feedback" accordingly [5, 6].

Currently, an autonomous wheelchair robot (PSUBOT) is being developed in the Department of Electrical Engineering of Portland State University (PSU). The robot is being developed primarily to transport a physically handicapped person within a known environment. The wheelchair robot is envisioned to use visual perception to "learn" its environment, recognize its bearings in the "learned" environment and use this perception as a navigational aid. A complete vision system would obviously involve real-time image capture, real-time processing of the images to derive pertinent image features and "recognize" the environment based on these features. To be particularly feasible for the autonomous wheelchair robot, the system must be implementable within the reasonable bounds of hardware requirements and constraints, real-timeliness and efficacy.

This thesis involves the development of the front end of the vision system, i.e., the design of an automated sequence of low to medium level image processing operations that will yield the pertinent image features for the scene recognition task. A brief history of mobile robots and a description of PSUBOT are given to show the constraints and conditions that surround the design development.

MOBILE ROBOTS

The idea of robots stretches back to the pre-electronic era when mechanical entities endowed with human-like behavior existed only in science fiction. The technological development really started in the late fifties when industrial robots were finally developed for some mechanical assembly, material transport and eventually material inspection in the manufacturing environment. The main objective was to alleviate use of human labor and increase production efficiency [7]. Robot research then was concentrated on mechanical manipulators either remotely controlled or preprogrammed to effect specific functions. Stanford Research Institute's SHAKEY marked the advent of mobile robot vehicles that are envisioned to be capable of autonomous navigation and real-time control of the robot system that interacts with the environment [4]. Autonomous robots then were required to have environment perception, planning and decision-making and real-time task execution control. Environment perception was mostly realized with the use of television cameras, rangefinders and proximity sensors to provide the robot with real-time information to perform path generation, guidance and target tracking. Because of the computation intensive characteristics of image processing, vision for perception was kept at a minimum. If ever vision was used actively, the system has to use multiple cameras for stereo vision and more powerful multiprocessors to cope with real-time requirements. Systems were developed for outdoor and indoor navigation but with complex and expensive hardware, primarily developed as research testbeds.

Applications for mobile robots have now bloomed from the automated factory to autonomous tanks, sentry robots, ship and aircraft applications in the military, patient and material transport in medical care services, mobile tractors and harvesting machines for agriculture, mobile lawn mowers and vacuum cleaners in the domestic front [8].

PSUBOT: THE AUTONOMOUS WHEELCHAIR ROBOT

PSUBOT is a motorized wheelchair robot being currently developed in the PSU Electrical Engineering Department. It is primarily intended to transport a physically handicapped person within a known environment particularly the inside of a building. The wheelchair is envisioned to be endowed with capabilities that might be lacking in its passenger. A handicapped passenger may have hand-eye coordination problems, may even be blind, or may just not be capable of operating or controlling the wheelchair. On these premises, the wheelchair must be capable of autonomous navigation. Autonomous in this context means that the wheelchair carries all the necessary equipment needed for navigation control to be able to move around the known environment freely, and must rely on sensory feedback to control its motion and find its way in the environment. The operations necessary for data processing of sensory feedback, path planning, decision making and hardware control have to be done onboard so that the motion will not be constrained by wires, communication links or distance from a controlling computer.

The wheelchair must have a vision system to show where it is and where it is going, an ultrasonic ranging system to ensure that its immediate path is clear of obstacles, a voice recognition system that will enable it to respond to verbal commands when needed, wheel feedback sensors and controllers to improve the straight line or trajectory motion. Global pathplanning is done by an intelligent database that keeps the map of the "learned" environment and the current state of the robot received from the subsystems. A central controller serves as the navigator, directing the wheel controller as appropriate based on information obtained from the intelligent database. Figure 1 shows the functional components of the PSUBOT.

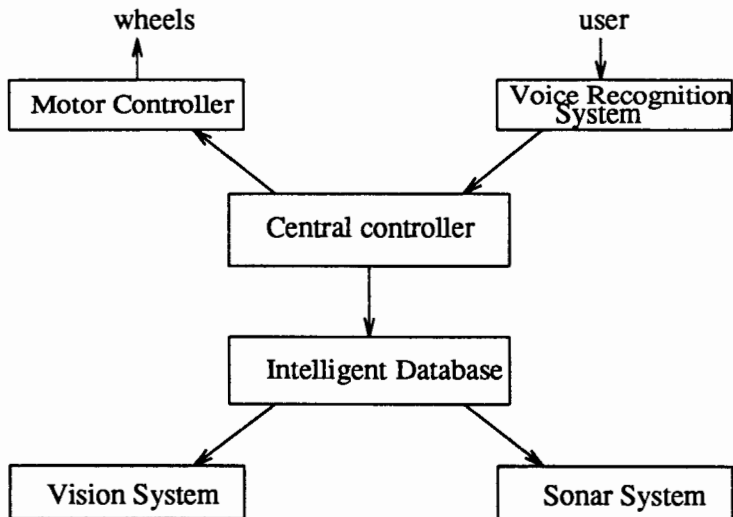


Figure 1. Functional components of the PSUBOT.

Currently, PSUBOT is voice controlled [9], i.e., navigational control is solely dependent on voice commands that may start or stop the motion of the wheelchair, direct forward or reverse motion, speed up or slow down smoothly and turn in a specified direction at a specified angle. A feedback system that uses speed sensing has been installed on each wheel, allowing the speed of each wheel to be monitored during the motion.

PROBLEM STATEMENT

So much work has been done to develop complex and expensive systems that the feasibility of their use in basic applications is outweighed by their complexity and cost [10]. This observation so aptly describes the developments in image processing and robotic technology when one thinks of using these developments in a simple indoor transport system like the PSUBOT. The fact that the robot is a wheelchair that must carry all its processors, controllers and power source in addition to its passenger to navigate about its environment is already a restriction on the size and type of processing hardware and data storage facilities that may be used. It is true that hardware technology can offer powerful, compact equipment and processors, but the cost of

such equipment makes it unreasonable for use on a wheelchair that is intended for practical use.

Since all the software processing will be done on board the vehicle and in real-time, all the processes will have to share whatever computer resources there are on board. This idea does not seem so attractive for an image processing system since image processing operations involve intensive calculations and intensive memory use. The vision system will also require storage of a lot of "learned" images that have to be on hand for the recognition task. The form by which the images are represented and stored could be a bottleneck in the memory use aspect and also in the ease or speed of the subsequent recognition task.

Thus, the design of the processing system must be such that there is a reasonable balance of simplicity, speed of computation, and data representation compactness in addition to optimum efficiency so as to meet the requirements and satisfy the constraints imposed on the entire robot system.

GOALS AND DESIGN DEVELOPMENT METHODOLOGY

The primary goal of the thesis is the development of a simple, inexpensive and automatic or unsupervised low to medium level image processing system that is directed towards the development of a vision system for the PSUBOT. At the current stage of conceptualization, the PSUBOT is primarily designed as an indoor transport system. As such, the vehicle is limited to the inside of a building where scenes would mostly consist of corridors, wall corners, door outlines, wall-floor boundaries and other fixtures common in indoor scenes. The image processing then may be limited to those operations that are primarily concerned with the extraction of scene features whose outlines may almost always be described as straight lines or some configurations of straight lines. To address the problem earlier stated, the primary goal may be broken down to the following subgoals: (1) selection of the appropriate processing techniques that will favor the extraction of straight line boundaries; (2) tailoring these techniques to suit system requirements and constraints (3) devising a straight line extraction scheme that will allow the scene

characterization to be represented in some compact form and which is almost ready for use by the scene recognition task; and (4) organizing the processes into a completely automatic sequence. An overview of the image processing system as a whole, and the hardware and software aspects of the low to medium level image processing system developed are presented in Chapter II.

Algorithms, techniques and ideas have swamped the field of image processing in the past years, each one advocating attractive results and effects. Comparative studies and evaluations of performance were published to give more light to the conditions for which the various methods are appropriate. The selection process was based on a balance of preserving image feature integrity as may be judged visually, simplicity and speed of calculations involved and amount of memory usage during the processing. Where ever applicable, published comparative studies and evaluations were used to guide the selection procedure.

To uphold the spirit of engineering, it was deemed reasonable to use available resources appropriate for the purpose, filling in the gaps where necessary rather than inventing new methods where the available ones will work as well. Found useful were Benjamin Dawson's Simple Image Processing Package(SIMPP2) [11, 12] and some programs developed by past students of the Department of Electrical Engineering of PSU. Some of the routines and programs which suited the purpose were used for the preprocessing and boundary extraction stages. These programs were, however, highly user interactive so that some interfaces have to be fixed to have a continuous processing. A program was developed to automatically determine the parameters originally supplied by the user to totally eliminate human intervention. Detailed descriptions of the preprocessing and boundary extraction stages are given in Chapters III and IV, respectively.

The most critical aspect of the development was the determination of an appropriate line feature extraction and characterization scheme since this processing stage determines how the scene is characterized for storage or for use by the recognition task. The results of this stage are practically the basis of the vision capability of the mobile robot. For reasons cited in Chapter V,

the Hough Transform method was finally chosen. However, a hierarchical approach was used to overcome the complications that are inherent to the method. This gave rise to the development of some data abstraction to simulate a pyramid structure so as to make the line extraction and characterization scheme be implementable in a PC. It must be noted that the working memory segmentation of a DOS-driven PC places severe constraints on the amount of active data space that may be used. The data structure also provided for the output of the characterization in terms of the hierarchical levels, a form which might prove useful for the scene recognition task. A complete discussion of the line feature extraction and characterization scheme is presented in Chapter V.

EVALUATION CRITERIA

The primary concerns in the design of an image processing system for an autonomous robot like the PSUBOT are: real-timeliness, efficacy of the image processing and transportability. Real-timeliness in the context of the wheelchair robot could be measured in terms of a reasonable percentage of the average speed of the wheelchair in an indoor environment, which if operated via a joystick by a passenger who has complete control of his senses, may be in the vicinity of 1 meter/sec. Efficacy may be measured in terms of how well each of the processing steps contributes to the quality of the scene features finally extracted and characterized in order to facilitate scene recognition based on these features. The goodness measure for the scene feature extraction is rather a qualitative one, i.e., it is based on how discernable the features are on the resulting processed image to a human viewer. An additional factor that may be considered within the domain of efficacy is the degree of noise tolerance of the image processing steps as this determines the amount of noise-eliminating steps that need to be used to yield credible features, precautionary steps that may weigh down on the overall processing time. Considering that the PSUBOT requires all of its subsystems to share processing facilities on board the vehicle, then all the image processing task must be implementable in the restricted confines of

the PC installed on the vehicle.

The implementation of an image processing system on a PC for a mobile robot that is designed for a practical purpose like passenger transport is a rather unexploited area in robotics. Though one such system (also a self-navigating wheelchair) was started in Arizona State University [13, 14], the image processing task was not so involved for use in scene recognition. Thus, the initial requirement of determining an effective combination of image processing steps that will allow scene recognition based on visual feedback, somehow shifts the prioritization of the evaluation criteria for this endeavor. Thus, the major concerns for this undertaking are transportability of the image processing system components and efficiency of scene feature extraction. Although processing speed was considered in all aspects of the development, it was not given the highest priority in the selection procedure, since the initial effort was concentrated at organizing a system that is transportable and efficient, a system that may serve as the base for future developments and improvements.

Whenever available, published performance evaluations and surveys were used to guide the selection procedure. However, the final choice of the techniques to be adopted and/or the parameters pertinent thereto were based on their suitability to the immediate goals, their effects on the other operations involved, and their final impact on the recognizability of the scene features extracted.

CHAPTER II

THE IMAGE PROCESSING SYSTEM

An image can be interpreted as an energy distribution over a surface. Imaging devices like cameras use image sensors that generate analog signals which represent the spatial energy variations in the image [15]. Sampling and quantization of these signals result in a digital image which has been discretized both in spatial coordinates and in brightness. The digital image is a two-dimensional array whose row and column indices identify a point in the image and the corresponding value of the array element (called pixel or picture element) represents the brightness of the image at that point. The pixel value called the gray level is a positive integer representing the quantized measurement of the brightness. Digital image processing involves the systematic treatment and manipulation of the pixel values so as to extract the desired information about the image.

HARDWARE ASPECTS

The image processing equipment basically consists of a video source that generates image signals, a frame grabber to digitize the image signals and store the digitized image in an image memory, and a computer to initiate and control the image acquisition process and perform other image processing functions on the digitized image.

The image processing system at hand uses a CCD camera as the video source, the PCVISIONplus FRAME GRABBER card by Image Technology Inc., and a PC-386SX connected as shown in Figure 2. An external video monitor may be optionally used to allow monitoring of the digital image during the processing. The details of how the components are interconnected for image acquisition is described in Appendix A.

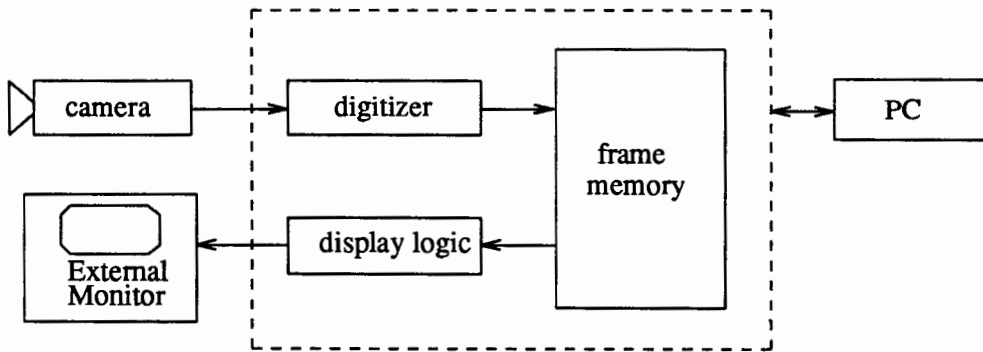


Figure 2. Hardware components of the image processing system.

The PCVISIONplus card plugs directly into an expansion slot in the PC. The card receives input video signal (RS170/330), digitizes the video signal to 256 gray levels and stores the resulting pixels in a 1024×512 byte frame memory [16]. The PCVISIONplus is controlled by the computer through 16 1-byte registers that are dedicated for use by the card. With the appropriate software interface, the computer initiates the image acquisition and may access the image memory in 64K blocks, allowing the computer to read or modify selected pixel positions or pixel values.

SOFTWARE ASPECTS

The image processing software consists of routines for initializing the PCVISIONplus registers, look-up tables (LUTS) and image memory, interfacing with the image memory, image acquisition, image enhancement, edge detection and extraction, and line feature extraction and characterization. The PCVISIONplus initialization routines were developed by past EE students [17] for the purpose of interfacing SIMPP (Benjamin Dawson's earlier version) with the PCVISIONplus. The image memory interface routines allow access to one or more pixels in the image memory for reading or modification. The image acquisition routine drives the frame grabber to digitize an input image and store the digitized image in the image memory. These

routines are parts of SIMPP2.

The image enhancement consists of Histogram Equalization and Image Smoothing. Histogram Equalization [1, 2] is supposed to improve contrast in preparation for the extraction of object boundaries whereas Image Smoothing [18] reduces extraneous data introduced by the acquisition hardware and the digitization process. The edge detection and extraction routines consist of the Edge Detection [19] using the Sobel operators to highlight object boundaries or edges; the Binarization [20] of the image to extract the detected edges; and the Thinning [21] of these edges to one-pixel-width lines. The threshold for the binarization is determined automatically using the Between-Class-Variance method [22]. The Labelling routine [11] is used prior to Thinning to eliminate small spots (surviving noisy pixels). Programs for Histogram Equalization, Image Smoothing, Edge Detection and Thinning were previously developed by Alvin Legate and Kelly Spiller in connection with a student project in the Department of Electrical Engineering of the Portland State University in Spring 1989. The Histogram Determination, Binarization and Labelling routines are parts of SIMPP2. The automatic Threshold Determination routine was developed in the course of this thesis to completely eliminate human supervision of the image processing. The above-mentioned routines were integrated into an automatic image-acquisition-to-edge-extraction processing sequence that produces an image that is ready for feature extraction.

The "Hierarchical Hough Transform" program developed in this thesis finally processes the results of the edge extraction stage to extract the lines that define the outlines of objects in the scene. The program characterizes these lines in terms of the normal parameters of the normal equation of the lines, the lines' midpoints and lengths which are then written in a file for the recognition stage. The lines are written in the order in which they occur in the hierarchical structure adopted, thereby allowing the recognition stage to exploit the classification offered by the structure.

All the routines and programs were written in C. The Borland C++ compiler (V1.0) and DOS 4.0 were used for the development on a PC-386SX.

AN OVERVIEW OF THE IMAGE PROCESSING SYSTEM

A picture of the vision system conceptualized for the PSUBOT is shown in Figure 3. It consists of two main components: the low to medium level image processing subsystem that will extract a description of the scene from the input images; and the scene recognition subsystem that compares the scene description generated by the first component with a previously "learned" scene. The low to medium level image processing system organized and developed in this thesis is intended to be the front end of the complete vision system for a mobile robot designed for autonomous navigation in a non-hostile environment, particularly the inside of a building.

Real-time image capture is implemented by digitizing a snapped image of the scene and storing the digitized frame in the image memory for further processing. After image acquisition, the image processing steps are divided into three major areas of concern: image enhancement,

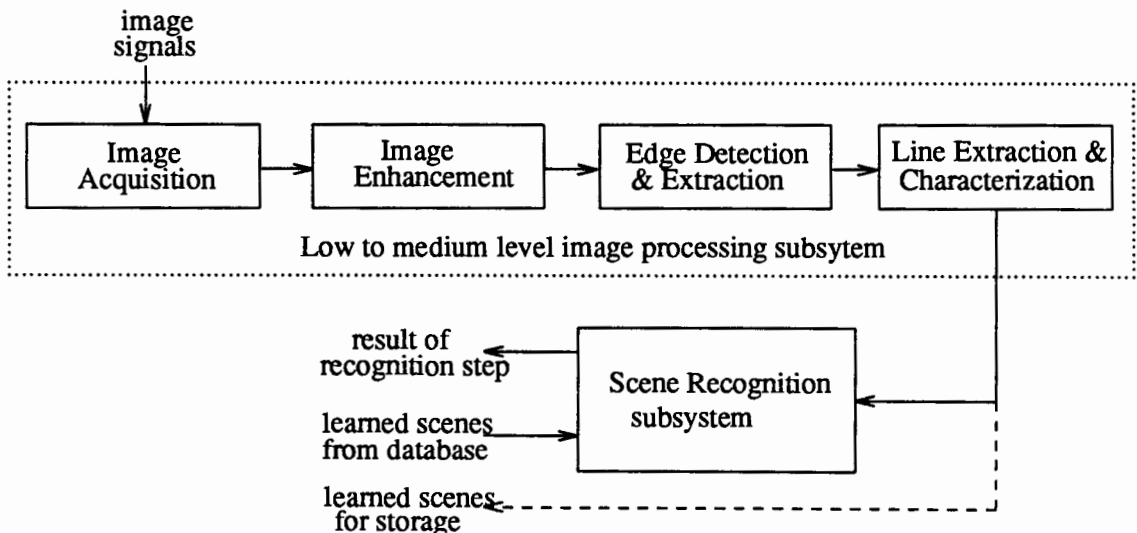


Figure 3. The vision system for the PSUBOT.

edge detection and extraction, and line extraction and characterization, all directed towards the characterization of indoor scenes in terms of line features that represent the outlines of the objects in the scene. Figure 4 shows the sequence of operations that comprise the low to medium level image processing subsystem.

Image enhancement is primarily intended to "improve" the quality of the digitized image so as to emphasize or deemphasize certain image characteristics. The image signals that are generated by the imaging device contain lots of physical information about the image, all chunked into what we call "brightness". Digitization of the analog image signals into a discrete number of brightness levels results in a digital image where neighboring pixels have gray levels that are almost similar or even the same. Also, the imaging and digitization processes could introduce extraneous values or noise into the array. Histogram Equalization of the digitized image spreads the pixel values to the maximum 256 grey level range. At the same time it enhances the contrast, thereby making the object boundaries more discernable. Use of the median filter gets rid of the spot noise without much effect on the boundaries. Edges are then enhanced or highlighted with the use of the Sobel edge operators and the image binarized to finally extract such edges.

In images of the inside of the building, corridors, lobbies, or rooms, most of the edges that mark the pertinent static features correspond to wall-floor boundaries, wall corners, wall-ceiling boundaries, door outlines, window outlines, and furniture outlines. These features are almost always straight lines or may be approximated with straight lines. To improve the characterization of such features, the extracted edges are thinned to one-pixel-width lines on which the Hough transformation is applied. The Hough transformation represents a line in terms of its polar parameters, thereby allowing the characterization of the line in terms of its polar equation. Based on the results of the Hough transformation, the scene of lines is characterized in terms of the length of the lines and their positions and locations in the image.

The hierarchical approach used in the line extraction affords a natural classification of the lines according to their relative expanse in the image. These allow characterization of the scene

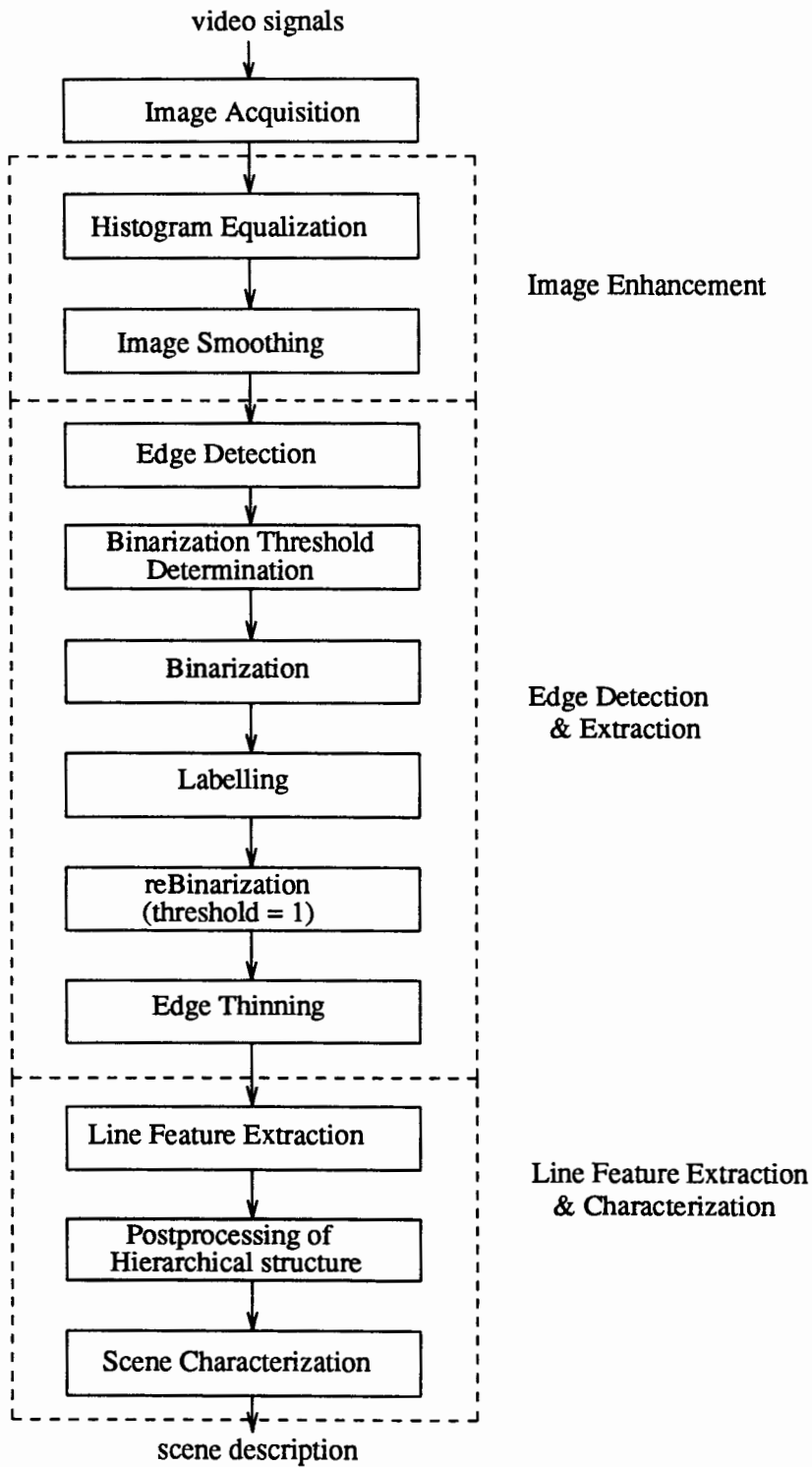


Figure 4. The low to medium level image processing steps.

in terms of lists of lines in each level of the hierarchy. A summary for each level contains the length of the longest line in the list, the total number of lines in the level, the number of horizontal lines, the number of vertical lines and the number of slanting lines. Each line in the list is attributed with its length, normal parameters with respect to the center of the image and its midpoint which is indicative of its location or relative position in the image. These characterizations are stored in the scene database for later use or are compared with those of the expected scenes to aid in the identification or confirmation of the robot's orientation, position or location.

The complete vision system interacts with the robot's intelligent database which initiates the operating mode for the image processing system and controls the camera position as it deems necessary to meet the needs of the central controller.

The vision system will have 3 modes of operation: a learn mode, where the system sets up the scene database as the robot is initially guided around the intended environment; a scene-identification mode where the system is to identify the bearings of the robot in a previously "learned" environment; and a navigation mode where the system is intermitently asked to confirm the bearings and provide the robot with the navigability of the path it is traversing.

The Learn Mode

The robot will be initially given a layout of the environment, the layout expressed in terms of a connectivity map, the rooms and hallway intersections as nodes and hallways as edges. The robot is then guided about the environment, and the image processing system is allowed to gather scene information to be associated with each node. In this mode, the system takes frontal views of the entrances to the rooms and intersections, and frontal views of the hallway on both sides of the entrances. Frontal snaps of the hallways will be taken at certain points along the hallway, and will be associated with the edge representing the hallway. This will serve as checkpoints that will be used during the navigation mode.

The Scene-identification Mode

The scene-identification mode will have two variations: initial identification of location in the "learned" environment; and identification of location with knowledge of previous bearings. In the first variation, snaps of the scene are compared with the scenes in the database until a scene is finally identified. The robot has to stop to get its initial bearings, and the database will stand attendant to the vision system and the sonar system to provide the need for possible scene models. In the second variation, snaps are taken until a particular snap matches a scene in a set of expected scenes in the vicinity, thereby confirming the robot's position and location. Being aware of the state of motion of the robot and the vicinity in which the robot is currently located, the database could note the time elements involved to guess the possible set that must be given to the vision system for confirmation.

The Navigation Mode

Navigability of a path requires that there exists sufficient space for the robot to move on. In this mode, the image processing system will be particularly partial to the scan of the lower portion of a frontal image, identifying lines corresponding to wall-floor boundaries that mark the hallway, giving the intelligent database some feedback regarding the freeness of the navigation path. This mode also makes use of the scene-identification mode to confirm the robot's relative position to a current target scene as deemed necessary by the intelligent database.

CHAPTER III

IMAGE ENHANCEMENT

Image enhancement techniques are primarily intended to prepare the image for further processing that will lead to the credible analysis and interpretation of desired image features. The techniques generally involve some transformations of the image gray levels so as to enhance or deemphasize certain image features. Image enhancement techniques are used to make the resultant image more suitable for specific applications, making the image enhancement step become problem-oriented [1]. Of particular interest are enhancement techniques that will prepare the image for the extraction of boundary features. Such enhancement requirement may be met by:

- (1) brightening the image by increasing the intensity level of each pixel by a constant value, an operation which may be required of images that are predominantly dark;
- (2) stretching the image so that chosen areas of particular interest are enlarged for more detailed analysis;
- (3) the use of histogram modification techniques to effect a global transformation in the gray level intensities according to a preset condition about the intensity distribution;
- (4) image smoothing to diminish noise introduced by spurious sampling and transmission effects;
- (5) sharpening operations to emphasize desired image features.

With processing speed and efficiency as the primary concerns, the image enhancement steps chosen are primarily aimed at transforming the image in such a way that objects become easily distinguishable from the background. Due to the limitations imposed by the image acquisition and digitization hardware used for the images taken for the tests, the images used consisted of 64 gray levels only. Histogram equalization is first applied to spread out the image

to the 256-gray-level intensity range and to improve contrast between objects in the image. The smoothing operation is then applied to eliminate noisy spots in the image. Object boundaries are finally sharpened in preparation for the boundary extraction steps.

The histogram equalization and smoothing operations are described in detail in the following sections, whereas the sharpening operation which is primarily concerned with the edge extraction is described in the next chapter.

HISTOGRAM EQUALIZATION

The gray level histogram of an image gives a global description of the image. Based on the gray level distribution, an image may be judged to have generally dark characteristics or the image has a predominance of light tones as the case may be. Histogram equalization is a technique that makes use of the gray level histogram to guide the gray level modification that will result into an image where the tones are more or less distributed uniformly. The technique is based on the transformation of the form:

$$s = T(r) \quad (3.1)$$

that produces a level s for every pixel value r in the original image. From elementary probability theory, if $T(r)$ is single-valued and monotonically increasing in the range of r , the probabilities for the s and r are equal [23], i.e.,

$$p_s(s) ds = p_r(r) dr \quad (3.2)$$

where $p_s(s)$ and $p_r(r)$ are the probability density functions for the s and r levels, respectively.

Histogram equalization makes use of a transformation function based on the cumulative distribution function [1], which indeed satisfies the single-valuedness and monotonicity requirements for a transformation function. The transformation function for mapping the original histogram to a new histogram of M gray levels is:

$$T(r) = M \int_0^r p_r(w) dw \quad (3.3)$$

where w is a dummy variable of integration. If $h(r)$ is the number of pixels for each gray level r of the original histogram H and $g(s)$ is the number of pixels for each gray level s of the new histogram G , then the probability density functions are expressed as:

$$p_s(s) = \frac{g(s)}{N}, \quad p_r(r) = \frac{h(r)}{N}. \quad (3.4)$$

Equation (3.2) may then be rewritten as:

$$g(s) ds = h(r) dr \quad (3.5)$$

which simply means that a small interval dr in the original histogram H is mapped to a corresponding interval ds in the transformed histogram G via the cumulative histogram as shown in Figure 5. The combination of equations (3.1), (3.3) and (3.4) produces:

$$s = M \int_0^r \frac{h(w)}{N} dw. \quad (3.6)$$

Direct differentiation of equation (3.6) gives:

$$\frac{ds}{dr} = \frac{M}{N} h(r)$$

which when substituted into equation (3.5) shows the new histogram G to have a constant frequency for any level s , i.e.,

$$g(s) = h(r) \frac{1}{\frac{M}{N} h(r)} = \frac{N}{M}$$

As a result, crowded gray level ranges get mapped to wider ranges and ranges where frequencies are low get mapped to narrower intervals as shown in Figure 5. The transformation results in an image where the contrast is enhanced in the more crowded range and the contrast is compressed in the less crowded range [2], thereby improving the overall visibility of the object boundaries. The technique also makes possible the transformation of an image from one gray level range to another desired gray level range.

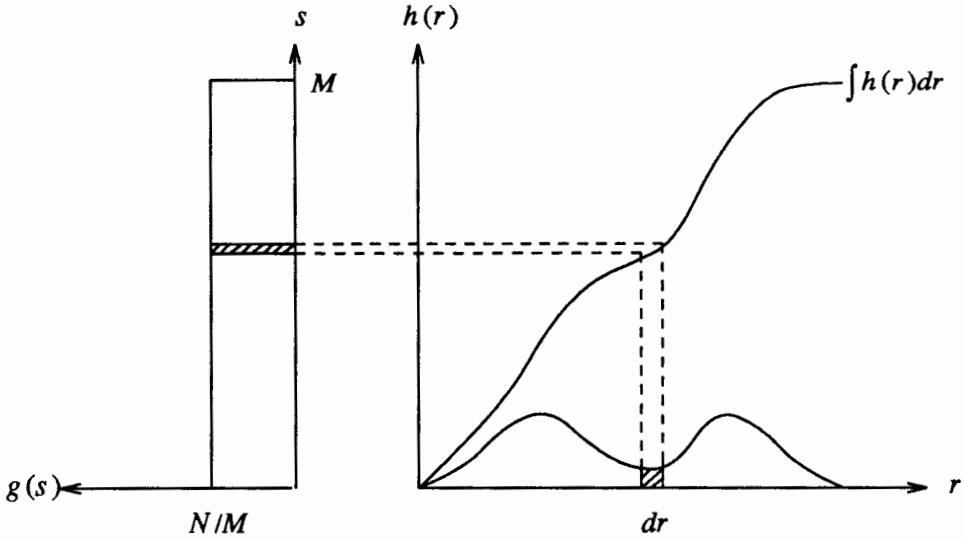


Figure 5. The mapping of a gray level histogram H to a uniform histogram G using the histogram equalization technique.

Implementation of Histogram Equalization on a Digital Image

The discussion presented above was based on the consideration of the gray level histograms as continuous functions. However, when gray levels assume discrete values as in the case of a digitized image, the gray level histogram of the digital image consists of counts (n_j) of pixels for each discrete gray level (r_j). For an image of N pixels, the probability density for each level is expressed as:

$$p_r(r_j) = \frac{n_j}{N}. \quad (3.7)$$

The new gray level s_k in the new M -gray-level range is then determined according to the discrete equivalent of equation (3.3) which is:

$$s_k = M \sum_{j=0}^k p_r(r_j). \quad (3.8)$$

The histogram equalization routine used calls the histogram determination routine for the generation of the gray level distribution $h(n_j)$ for $j = 0$ to $j = M - 1$. The probability density is then determined for each gray level (r_j) according to equation (3.7). A look-up table of the new

gray level distribution based on equation (3.8) is then created for $k = 0$ to $k = M$. Each pixel value r in the input image is then replaced by the corresponding s value from the look-up table, resulting in the transformed image.

Figure 7 shows the effect of histogram equalization on a test image shown in Figure 6. It is obvious that the resulting image has better contrast than the input test image. The gray level histograms in Figure 8 show that the test image has only 64 gray levels, whereas the transformed image is spread out over the 256-gray-level range. The histogram of the transformed image is however not flat as described earlier since the gray levels are discrete.

IMAGE SMOOTHING

The sampling system and the transmission channel used in the image digitization process can introduce extraneous data into the acquired digital image. These spurious data often appear as very low or very high intensity levels for some pixels, making the pixels inconsistent with their neighbors. The smoothing techniques are intended to diminish if not totally eliminate these spots so that they do not interfere in the operations that identify image features based on intensity variations in pixel neighborhoods. Smoothing operations may be effected either in the spatial domain or in the frequency domain. Spatial domain techniques make use of the intensity levels in a pixel's neighborhood to adjust the pixel's intensity level so that it becomes consistent with its immediate neighbors. On the other hand, frequency domain techniques implement the operation by the convolution or multiplication of the Fourier transform of the image with a low pass filter function. In general, the operation attenuates the high frequency components of the Fourier transform spectra based on some threshold that is characteristic of the filter function used. Sharp transitions in the gray levels of the image which occur around noise spots and edges heavily contribute to the high frequency components of the image's Fourier transform. The convolution results in blurring the pixels that contribute to the high frequency components.

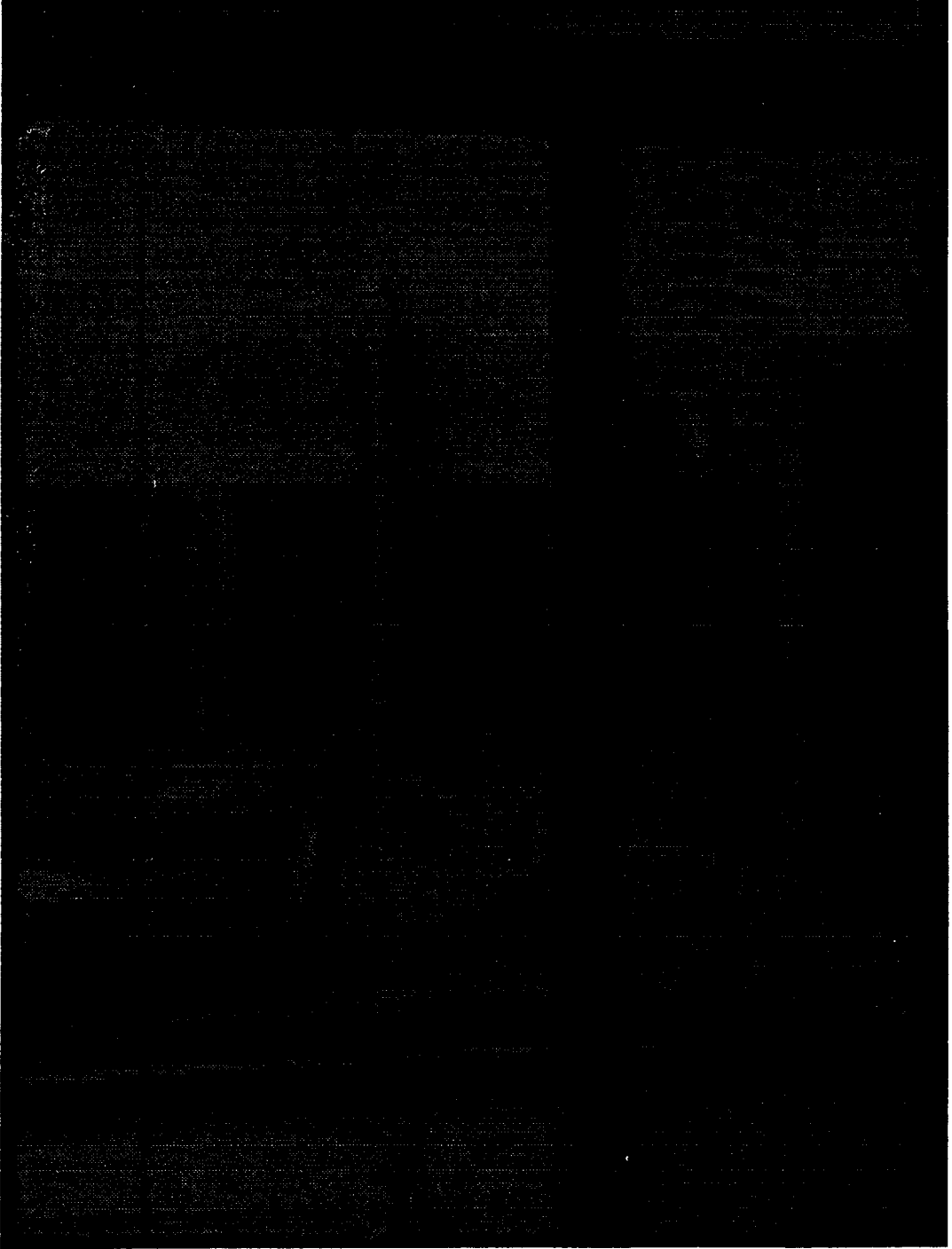


Figure 6. A test image.



Figure 7. The test image after histogram equalization.

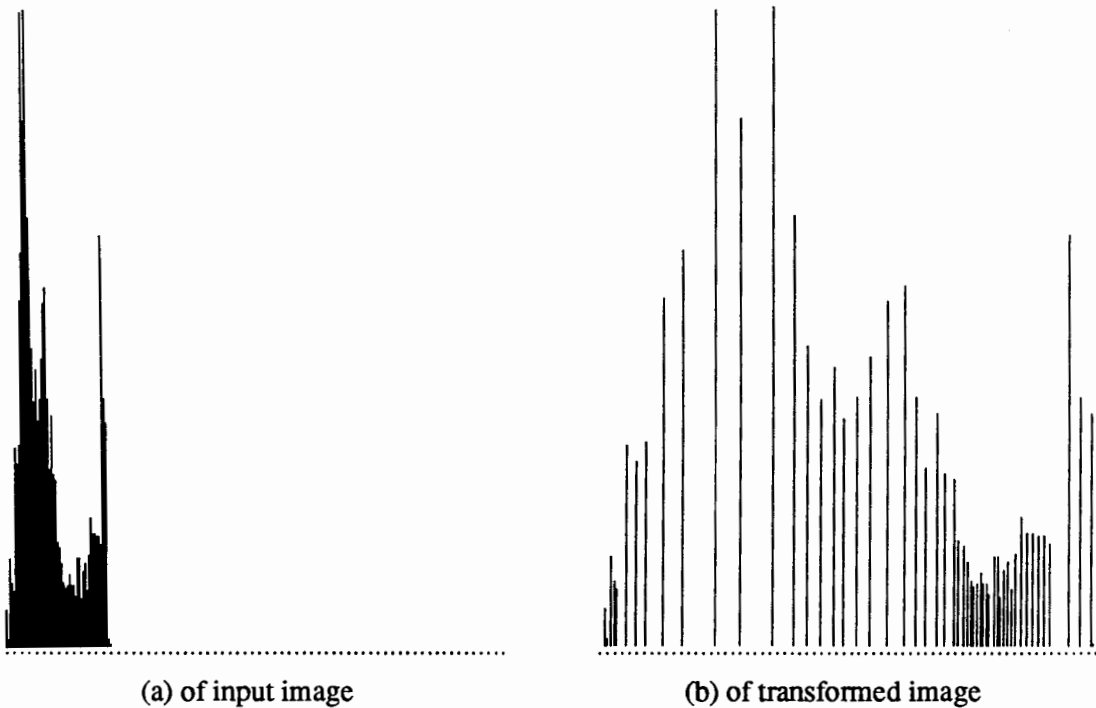


Figure 8. The gray level histograms of the images in Figures 6 and 7.

The need for transforming the image function into its Fourier transform, performing the convolution in the transform domain and inverting the resultant transform back to the original domain mean a lot of calculations unless specialized hardwares are used to implement the transformation. The spatial domain techniques are much simpler and more practical for real-time applications if no special hardware is to be considered and if calculation simplicity can be equated to speed without considerable loss of the efficacy. Also, the spatial domain techniques do not need preset threshold values for the pixel value transformation, making the techniques self-sufficient and not dependent on previous knowledge of some appropriate thresholds.

The spatial domain techniques like neighborhood averaging and median filtering base the transformation of the intensity level of every pixel (x, y) on the intensity levels of the pixels in a predefined neighborhood of (x, y) . In neighborhood averaging, a pixel's intensity level $I(x, y)$ is replaced by the average of the intensity values of the pixels in its neighborhood, i.e.,

$$I(x, y) = \frac{1}{M} \sum I(j, k), \quad (j, k) \in S$$

where S is the set of coordinates of the pixels in the neighborhood of (x, y) , including (x, y) itself, and M is the total number of pixels in the neighborhood. On the other hand, median filtering replaces $I(x, y)$ by the median intensity value in the neighborhood:

$$I(x, y) = \text{median} \{ I(j, k) : (j, k) \in S \}.$$

One of the principal difficulties of the neighborhood averaging method is its blurring effect on edges and other sharp details [18] which makes it undesirable for use in images where edges are of primary interest. As Bovik, Huang and Munson [24] reiterated about smoothing techniques: "if noise reduction is to be effected prior to detecting edges, then the filtering strategy used must not severely degrade the edge content of the image", it is indeed reasonable that the smoothing technique must not sacrifice edge integrity for noise suppression. Their study on the efficacy of median filters showed that the technique is indeed effective in removing noise from images while retaining the integrity of the edges. They also showed that the median filters improve the performance of both the zero-crossing edge operator and the conventional gradient-based edge, operator which they used for the edge detection. Of the different median filter geometries they tested (see Figure 9), the square geometry performed best in terms of noise suppression and edge preservation which they attributed to the larger span of the filter.

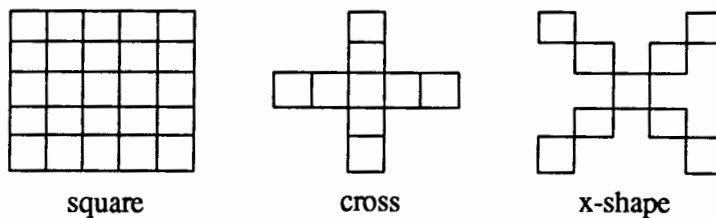


Figure 9. Median filter geometries used by Bovik, Huang and Munson.

Implementation of the Median Filtering Technique

The median filtering technique was chosen to effect smoothing prior to edge detection and extraction. As was pointed out earlier, the technique was shown to be effective in noise suppression and edge preservation. A 3×3 neighborhood window was used for each pixel position. The filtering was done by centering the window on each pixel, the pixel intensity values within the window sorted to locate a median value, and the center pixel's value replaced by this value. The application of the median filtering technique to an image may be described by the following pseudocode:

```

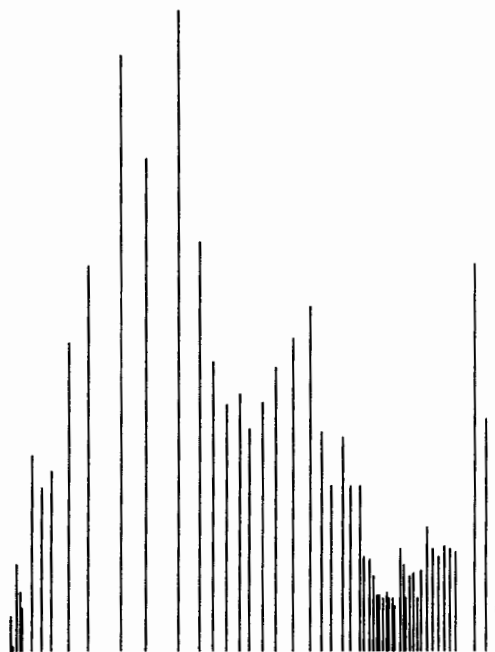
/* median filtering : 3x3 filter window */
/* begin */
  read first 3 image rows into buffer
  while (last buffer row has an image row)
  { start window at first 3 columns of buffer
    while (window within buffer)
    { insert sort each pixel value into sorting array
      write middle value in sorting array into center pixel in image
      shift window one pixel column along buffer
    }
    shuffle buffer rows (shift out top row, shift up rows 2 and 3)
    read the next image row into row 3 of buffer
  }
/* end */

```

Figure 10 shows the effect of median filtering on the image in Figure 7. The operation indeed eliminated some relatively bright spots in the original image yielding a much smoother resulting image. The resulting gray level histogram reflects the decrease in the pixel counts at some higher gray levels and some increase at the lower gray levels.



(a) the smoothed image



(b) histogram of the smoothed image

Figure 10. Effect of median filtering on the image in Figure 7.

CHAPTER IV

EDGE DETECTION AND EXTRACTION

Object boundaries or edges are characterized by abrupt changes in the gray levels of neighboring pixels. Edge detection techniques exploit this property to identify and enhance or sharpen the pixels along the object boundaries with the use of some appropriate edge operators. The sharpened edge pixels are then extracted by the binarization of the image, i.e., based on an appropriate gray level threshold value, the image is transformed into a two gray-level representation so that the edge pixels are assigned one gray level and the non-edge pixels are assigned the other gray level. The extracted edge pixels are then further thinned to one-pixel widths in preparation for a higher level of abstraction of the edges which is the subject of the next chapter.

EDGE DETECTION

Edge detection involves the application of edge operators to small areas in the image to identify the pixels that comprise possible edge boundaries. An edge operator or detector is a mathematical operator or its equivalent, so designed to detect the presence of a local edge in a small area of the image based on the variation of gray levels in the pixels comprising the area.

Edge operators fall into 3 main classes [2]:

- (1) operators that approximate the mathematical gradient;
- (2) template matching operators that use multiple templates for different possible orientations of the edge;
- (3) operators that fit local intensities with parametric models.

Despite the development of more sophisticated edge detectors, the simpler gradient operators such as the Robert's Cross, Prewitt and Sobel operators are well established and commonly

used, chiefly because of their computational simplicity which is an important consideration in robot-oriented vision [25].

The Gradient

In the realm of continuous two-dimensional functions, the gradient of $f(x, y)$ which is denoted as $\nabla f(x, y)$ in equation (4.1), is defined as the vector that points in the maximum rate of increase of the function $f(x, y)$. Its magnitude which is denoted as $G(x, y)$ in equation (4.2), is the maximum rate of increase of $f(x, y)$ per unit distance in the direction of the vector, the direction as given in equation (4.3). To simplify notations, the symbols G_x and G_y are used to denote the partial derivatives:

$$G_x = \frac{\partial f(x, y)}{\partial x}, \quad G_y = \frac{\partial f(x, y)}{\partial y}$$

$$\text{gradient vector: } \nabla f(x, y) = \begin{bmatrix} G_x \\ G_y \end{bmatrix} \quad (4.1)$$

$$\text{gradient magnitude: } G(x, y) = \left| \nabla f(x, y) \right| = (G_x^2 + G_y^2)^{1/2} \quad (4.2)$$

$$\text{gradient direction: } \alpha = \tan^{-1} \frac{G_x}{G_y} \quad (4.3)$$

When operating on discrete quantities as in digital images where x, y and $f(x, y)$ are natural numbers, the partial derivatives are approximated by the finite differences in the orthogonal directions x and y as:

$$\begin{aligned} G_x &= f(x, y) - f(x+1, y) \\ G_y &= f(x, y) - f(x, y+1) \end{aligned} \quad (4.4)$$

where the relationships between the pixels are shown in Figure 11(a). The gradient magnitude approximation may be further simplified as the sum of the absolute values of G_x and G_y , a form which is more desirable for computer implementation of the gradient [1]:

$$G(x, y) = \left| G_x \right| + \left| G_y \right| \quad (4.5)$$

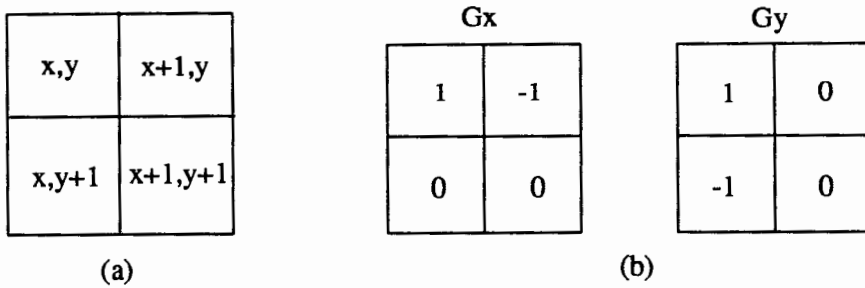


Figure 11. The 2×2 convolution masks for the digital derivative approximation. (a) Mapping of a 2×2 pixel neighborhood, (b) the digital derivative convolution masks.

The gradient magnitude as approximated according to equation (4.5) may be noted to be proportional to the difference in gray levels between adjacent pixels. Thus, the magnitude is relatively large where pixel values change abruptly, is small in the regions that are fairly smooth, and is zero where the gray level is constant.

The digital derivatives G_x and G_y of the equation set (4.4) are more conveniently represented as convolution masks as shown in Figure 11(b). This simplifies the digital derivative approximation as the convolution of the 2×2 neighborhoods of the image array with the masks. In the context of pixel areas and masks, convolution is a weighted summation process, with each element of the mask representing the weighting factor for the value of the corresponding pixel. Thus, the convolution result is simply the sum of the products of the pixel values in the image area and the corresponding weighting factors in the mask. The gradient magnitude may then be approximated according to equation (4.5) using the convolution results.

A common implementation of the gradient method described above is the generation of the so-called gradient image $g(x, y)$ [1]. A gradient image $g(x, y)$ results when the value of the gradient magnitude of the pixel in the $f(x, y)$ image is assigned to a corresponding pixel in the $g(x, y)$. The gradient image pixels that correspond to the edge pixels would be prominently bright, whereas those corresponding to the non-edge pixels tend to be darker.

The Gradient Edge Detectors

Several edge operators were developed to approximate the gradient in attempts to accurately mark pixels that comprise the boundaries. A number of these operators are described in [19] and [25]. These edge operators (also commonly referred to as differential operators) use convolutions to approximate the x and y components of the image intensity gradient. Most notable of the said operators are the Robert's Cross, Prewitt and Sobel operators as advocated by Kitchen and Malin [25], Ballard and Brown [2] and Levialdi [19]. The convolution masks for these operators are shown in Figure 12.

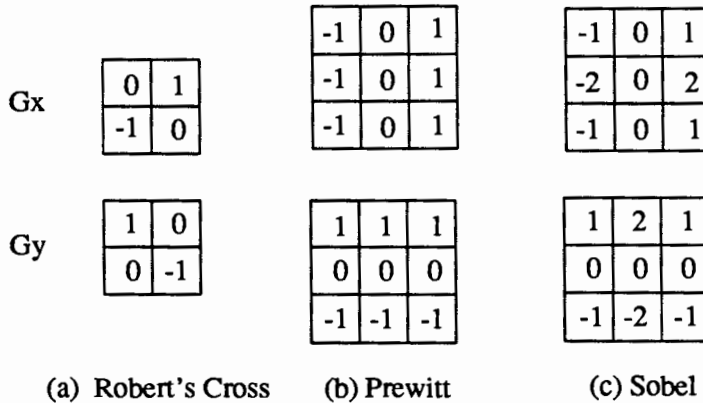


Figure 12. Differential edge operators.

The Robert's Cross operator uses the cross differences in the 2×2 neighborhood whereas the Prewitt and Sobel operators approximate the gradient in 3×3 neighborhoods. The Prewitt and Sobel operators introduce local averaging to reduce the effect of noise, a factor that Ballard and Brown attributed to their better performance over the Robert's Cross operator. The Sobel operator is very similar to the Prewitt operator with a heavier weighting assigned to pixels that are closer to the center of the neighborhood, a factor that was suspected to account for the superior performance of the Sobel operator over the other operators in comparative studies made by the above-mentioned authors. Levialdi also emphasized that the Sobel operator performed

sufficiently well even on vertical and diagonal edges on which white noise was added with noise to signal ratios of 1 and 10.

Sobel Operator. The Sobel operator was chosen for the system at hand because of its advocated superior performance in addition to the computational simplicity inherent to convolution-type operators. The Sobel operator approximates the gradient in a 3×3 pixel neighborhood by the convolution of the masks in Figure 12(c) with the 3×3 neighborhood of the center pixel x, y shown in Figure 13. The convolution results into the following digital derivative expressions:

$$G_x = [f(x+1,y-1) + 2f(x+1,y) + f(x+1,y+1)] - [f(x-1,y-1) + 2f(x-1,y) + f(x-1,y+1)]$$

$$G_y = [f(x-1,y-1) + 2f(x,y-1) + f(x+1,y-1)] - [f(x-1,y+1) + 2f(x,y+1) + f(x+1,y+1)]$$

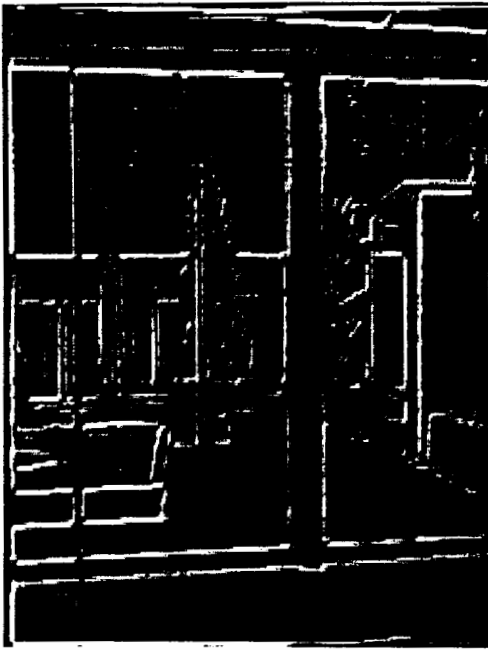
$x-1,y-1$	$x,y-1$	$x+1,y-1$
$x-1,y$	x,y	$x+1,y$
$x-1,y+1$	$x,y+1$	$x+1,y+1$

Figure 13. Mapping of the 3×3 pixel neighborhood.

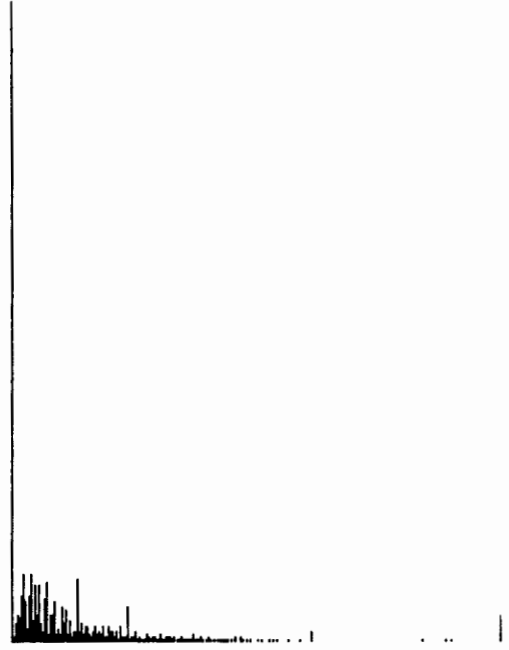
The magnitude of the gradient approximated according to equation (4.4) is then assigned to the center pixel x, y of the gradient image $g(x, y)$, i.e.,

$$g(x, y) = \left| G_x \right| + \left| G_y \right| .$$

Effecting the convolution for each pixel in the image results in an image where pixels in the vicinity of the boundaries are highlighted because of the higher gradient magnitudes at these points. Figure 14 shows the effect of the edge detection with the use of the Sobel operator on the image in Figure 10(a).



(a) gradient image



(b) gray level histogram of (a)

Figure 14. The gradient image of Figure 10(a).

BINARIZATION

The edge pixels identified and enhanced by the edge detection step must be extracted from the gradient image in order to focus the subsequent transformations on these regions which indicate the object boundaries. A popular technique in extracting objects of interest from an image is thresholding segmentation, or binarization as it is commonly called. The technique involves a selection of a threshold gray level value with which the image will be transformed into a 2-gray-level representation, i.e.,

$$f_t(x, y) = \begin{cases} b_0 & \text{if } f(x, y) \geq T \\ b_1 & \text{otherwise} \end{cases}$$

where $f(x, y)$ is the pixel's gray level value in the input image, $f_t(x, y)$ is its resulting gray level in the binary image consisting of gray levels b_0 and b_1 , and T is the threshold gray level. The threshold T must be so selected so that it discriminates the pixels that belong to the object

desired from the pixels that are not of interest, henceforth referred to as belonging to the background based on the pixel gray levels. Selection of the threshold may be based on experience with a set of selected typical images or from some previous knowledge of the gray levels of the object pixels or may be based on the behavior of the gray level histogram of the image.

AUTOMATIC THRESHOLD SELECTION

Basically there are two different approaches for thresholding: local thresholding and global thresholding. In local thresholding, the image is subdivided into subimages and the threshold for each subimage is determined based on the local properties of the subimage. Global thresholding selects a single threshold value and applies this value to the entire image. Automatic global thresholding techniques make use of the gray level histogram as the basis of the threshold selection. If the object is clearly distinguishable from the background, the histogram will be bimodal, i.e., there will be two peaks, each peak representing the gray level concentration of one group, and the bottom of the valley between the peaks representing the separation between the two groups. However, in most images, the histogram is not strictly bimodal or the peaks may vary significantly in size or the valley may be relatively wide, thereby complicating the location of the point of separation between the object and the background [26]. Histogram modification methods have been suggested by Weszka and Rosenfeld [27] depending on the histogram behavior. However, examining every histogram, deciding on a modification method to use and finally deciding on the threshold is a cumbersome procedure if the histogram behaviors vary widely as in images of scenes.

To overcome the difficulties of locating the bottom of the valley, Pun [28] developed the so-called entropic method that locates the threshold based on the entropies of the object and background groups as could be deduced from the histogram. Entropy is a measure of the degree of randomness of a set of random variables. Assuming that the probabilities of occurrence of each gray level are statistically independent, Pun developed a criterion measure based on the a

posteriori entropy of the histogram. The gray level at which the measure is maximum is supposed to be the threshold. Kapur, Sahoo and Wong [26] later found some derivation flaws in Pun's criterion measure which they rectified. They came up with their version of the entropic method by deriving two probability distributions from the original histogram: one for the object group, the other for the background group. The sum of the entropies of the two distributions becomes the criterion measure, and the gray level at which the criterion measure achieves the maximum is selected as the threshold. (The formulations for the method will be given later). However, they also showed that depending on the behavior of the histogram, the criterion measure plot may have more than two peaks, thereby requiring the final selection of the threshold to be made visually. An improvement of the entropic thresholding method proposed by Abutaleb [29] makes use of two-dimensional entropy. Two-dimensional entropy involves adding the spatial gray level distribution to the one-dimensional gray level distribution. The gray level of each pixel and the average gray-level value of its neighborhood are taken into consideration for the entropic threshold. However, Abutaleb himself found that the one-dimensional method (described earlier) yields comparable results with the two-dimensional method especially with noisy images, making the one-dimensional method preferable over the more computationally expensive two-dimensional method.

Another approach that determines the global threshold automatically and is not concerned with the bottom of the valley is the Between-Class-Variance method devised by Otsu [22]. The method is based on discriminant analysis. It involves a partitioning of the pixels into the object and background classes and determines the threshold based on some class variance criteria. The method is computationally simpler than the entropic method since the entropic method uses the logarithm function whereas variance computations simply involve squares.

A survey of thresholding techniques conducted by Sahoo, Soltani and Wong [30] showed both methods to perform well in terms of their shape and uniformity measures with bimodal images. However, the Between-Class-Variance method outranked the entropic method and all

the other methods considered in test cases that do not have bimodal histograms. The results of the survey are consistent with the comparative performance study of global thresholding techniques made by Lee and Chung [31] which also showed the Between-Class-Variance method to perform well regardless of object size. The entropic method worked well with varying object sizes provided the distributions of the object and the background are similar. Both methods however were found to be sensitive to noise. Both surveys ranked the Between-Class-Variance method as the best global thresholding method based on their overall analyses of the strengths and weaknesses of the different methods.

In the light of the foregoing analysis, the Between-Class-Variance method seems to be the better threshold selector for the binarization of the gradient image, considering that: (1) the method makes an unsupervised determination of the threshold; (2) the edge pixels that comprise the object group is indeed much smaller than the background group; (3) there is no assurance that the image to be binarized will always be bimodal, nor the distributions of the object and the background will always be similar; and (4) the method is computationally simpler than the entropic method or any other global thresholding method considered in the above mentioned surveys.

The Entropic Method

The improved entropy-based method developed by Kapur, et.al., treats the probability distribution of the gray levels as two distributions, one for the objects, the other for the background. Let n_i be the number of pixels of gray level i in an image of N pixels and $M+1$ gray levels. The gray level histogram may then be treated as a probability distribution, i.e., with p_i as the probability of occurrence of gray level i :

$$p_i = \frac{n_i}{N}, \quad p_i \geq 0, \quad \sum_{i=0}^M p_i = 1.$$

Two probability distributions are defined from the probability distribution of the gray levels

described above, distribution A for the discrete values 0 to s , and distribution B for $s+1$ to M as follows:

$$A: \frac{p_1}{P_s}, \frac{p_2}{P_s}, \dots, \frac{p_s}{P_s}$$

$$B: \frac{p_{s+1}}{1-P_s}, \frac{p_{s+2}}{1-P_s}, \dots, \frac{p_M}{1-P_s}$$

where: $P_s = \sum_{i=0}^s p_i$.

To simplify notations in the formulations, let:

$$H_s = - \sum_{i=0}^s p_i \ln p_i$$

$$H_M = - \sum_{i=0}^M p_i \ln p_i$$

where: \ln is the natural logarithm.

The entropies for the distributions are:

$$H(A) = - \sum_{i=0}^s \frac{p_i}{P_s} \ln \frac{p_i}{P_s} = \ln P_s + \frac{H_s}{P_s},$$

$$H(B) = - \sum_{i=s+1}^M \frac{p_i}{1-P_s} \ln \frac{p_i}{1-P_s} = \ln(1-P_s) + \frac{H_M - H_s}{1-P_s}.$$

The entropy of the complete probability distribution is the sum $\psi(s)$ of $H(A)$ and $H(B)$:

$$\psi(s) = \ln P_s (1-P_s) + \frac{H_s}{P_s} + \frac{H_M - H_s}{1-P_s}, \quad (4.6)$$

$\psi(s)$ is maximized to obtain the maximum information between the object and the background distributions in the image while the value of s that maximizes $\psi(s)$ is the threshold value. The maximum $\psi(s)$ occurs when the distributions A and B are identical or similar.

The Between-Class-Variance Method

Otsu [22] formulated the Between-Class-Variance Method based on discriminant analysis, establishing a criterion for evaluating the "goodness" of threshold that leads to the automatic

selection of an optimal threshold. The formulation is as follows:

The gray-level histogram is regarded as a probability distribution:

$$p_i = \frac{n_i}{N}, \quad p_i \geq 0, \quad \sum_{i=0}^M p_i = 1. \quad (4.7)$$

where n_i is the number of pixels of gray level i in an image of N pixels and $M+1$ gray levels.

Let a threshold level k classify the pixels into two classes C_0 and C_1 corresponding to the object and the background. C_0 will be composed of pixels with levels 0 to k and C_1 will be composed of those with levels $k+1$ to M . Let $\omega(k)$, $\mu(k)$ and μ_T respectively denote the zeroth-order and the first-order cumulative moments of the histogram, and the total mean level of the image to be subjected to the threshold:

$$\begin{aligned} \omega(k) &= \sum_{i=0}^k p_i \\ \mu(k) &= \sum_{i=0}^k ip_i \quad \mu_T = \mu(M) = \sum_{i=0}^M ip_i \end{aligned} \quad (4.8)$$

The probabilities of class occurrence are:

$$\begin{aligned} \omega_0 &= \Pr(C_0) = \omega(k) \\ \omega_1 &= \Pr(C_1) = 1 - \omega(k) \end{aligned} \quad (4.9)$$

The class mean levels are:

$$\begin{aligned} \mu_0 &= \sum_{i=0}^k i \Pr(i | C_0) = \sum_{i=0}^k i \frac{p_i}{\omega_0} = \frac{\mu(k)}{\omega(k)} \\ \mu_1 &= \sum_{i=k+1}^M i \Pr(i | C_1) = \sum_{i=k+1}^M i \frac{p_i}{\omega_1} = \frac{\mu_T - \mu(k)}{1 - \omega(k)} \end{aligned} \quad (4.10)$$

For any choice of k , the zeroth-order and first order moments are related by:

$$\omega_0 \mu_0 + \omega_1 \mu_1 = \mu_T, \quad \omega_0 + \omega_1 = 1. \quad (4.11)$$

The class variances are:

$$\begin{aligned} \sigma_0^2 &= \sum_{i=0}^k (i - \mu_0)^2 \Pr(i | C_0) \\ \sigma_1^2 &= \sum_{i=k+1}^M (i - \mu_1)^2 \Pr(i | C_1) \end{aligned} \quad (4.12)$$

The within-class variance σ_W^2 , the between-class variance σ_B^2 and the total variance σ_T^2 of the levels are given by:

$$\begin{aligned}\sigma_W^2 &= \omega_0 \sigma_0^2 + \omega_1 \sigma_1^2 \\ \sigma_B^2 &= \omega_0 (\mu_0 - \mu_T)^2 + \omega_1 (\mu_1 - \mu_T)^2 \\ \sigma_T^2 &= \sum_{i=0}^M (i - \mu_T)^2\end{aligned}\quad (4.13)$$

To evaluate the "goodness" of the threshold at level k the following discriminant criterion measures or measures of class separability were considered:

$$\lambda = \frac{\sigma_B^2}{\sigma_W^2}, \quad \kappa = \frac{\sigma_T^2}{\sigma_W^2}, \quad \eta = \frac{\sigma_B^2}{\sigma_T^2}.\quad (4.14)$$

Motivated by the conjecture that well thresholded classes would be separated in gray levels, and conversely, a threshold giving the best separation of classes in gray levels would be the best threshold, the problem can be reduced to the search for a threshold k that maximizes one of the criterion measures in equation (4.14). Since $\sigma_W^2 + \sigma_B^2 = \sigma_T^2$, the discriminant criteria that maximizes any of the above measures for k are equivalent to one another. It is noticeable that the measures λ and κ are functions of class variances (second-order statistics), whereas η is a function of the class means only. Also, σ_T^2 is independent of k , which means that the threshold level k that maximizes η equivalently maximizes σ_B^2 . Thus, η is the simplest of the above measures for the evaluation of the goodness of the threshold at level k . By using the relation from (4.11) in (4.13), the formula for σ_B^2 simplifies to:

$$\sigma_B^2 = \omega_0 \omega_1 (\mu_1 - \mu_0)^2.\quad (4.15)$$

Substitution of (4.9) and (4.10) into equation (4.15) finally expresses σ_B^2 in terms of the simple cumulative quantities in (4.8):

$$\sigma_B^2(k) = \frac{[\mu_T \omega(k) - \mu(k)]^2}{\omega(k)[1 - \omega(k)]}.\quad (4.16)$$

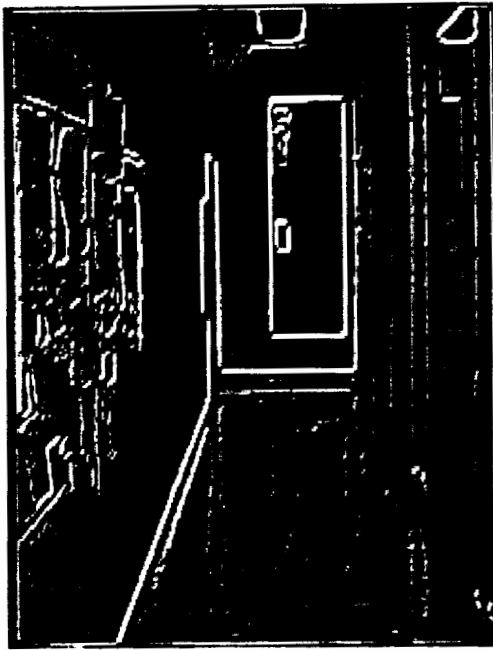
Equation (4.16) is always defined within the effective range of the histogram, i.e., $\omega(k)[1 - \omega(k)] > 0$ or $0 < \omega(k) < 1$. Equation (4.15), shows that σ_B^2 takes the minimum value

of zero when $\omega(k) = 0$ or 1 making all pixels either C_0 or C_1 and takes a positive and bounded value for any k within the effective range of the histogram. The selection of the optimal threshold k^* is then reduced to a sequential search for a k within the effective range of the histogram that maximizes the between-class variance σ_B^2 .

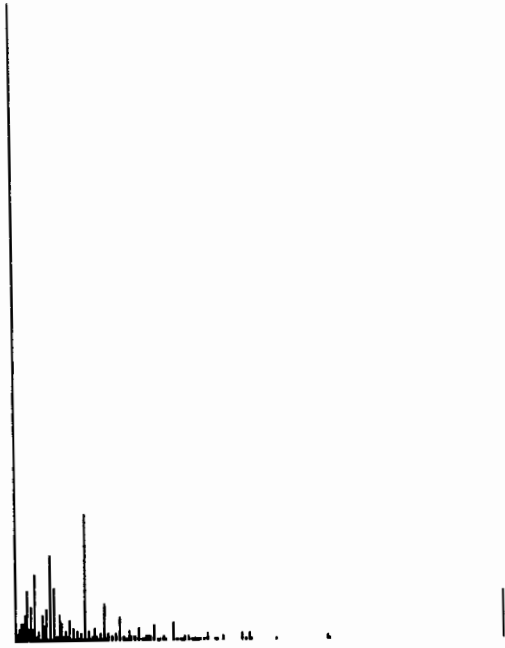
Application to the test scene images

For the sake of confirming the better thresholding method for the images for which this system is developed, both the entropic method by Kapur, et.al. and the Between-Class-Variance (BCV) method of Otsu were tried on the test images. The C program source codes written for this purpose are presented in Appendices B and C for the Between-Class-Variance method and the entropic method, respectively. For both methods the effective range of the gray level histogram was first determined so as to restrict the search within that range only. Tests were made on two treatments of the gray level range: one where all the gray levels were considered (start = 0), another when only the non-zero gradients were taken into account (start > 0). It must be noted that the major objective is to find a gradient magnitude value (the pixel values in the gradient image) that will classify the pixels along the boundaries as the edge pixels. A zero gradient magnitude denotes that the pixel is definitely far from the boundary.

The binarization thresholds for the image in Figure 15(a) are tabulated in Table I and the resulting binarized images are shown in Figure 16. The results showed that the entropic method consistently gave much lower thresholds than the BCV method, resulting in much thicker edges. It is also noted that excluding the zero-gradient pixels from the classification resulted in higher values for the criterion measures, implying the better separability of the groups. As expected, lower binarization thresholds give thicker edges, the edges getting thinner as the threshold is increased. Thinner edges are favored as long as no considerable edges or parts of the edges are lost as will be discussed in the next section. Timed runs on a PC-386SX showed that the BCV method takes approximately 50 msec to determine the threshold for an image, while the entropic



(a) gradient image



(b) gray level histogram of (a)

Figure 15. The gradient image to be binarized.

TABLE I

BINARIZATION THRESHOLD DETERMINATION RESULTS

test image	histogram range	entropic method		BCV method	
		threshold	$\max\psi$	threshold	$\max\eta$
1	0 - 255	68	8.1403	86	0.72191
	1 - 255	75	8.7958	93	0.72573
2	0 - 255	43	7.1009	94	0.72637
	1 - 255	51	7.9652	107	0.73562
3	0 - 255	47	7.3037	111	0.77228
	1 - 255	53	8.0879	120	0.79299



(a) threshold = 107



(b) threshold = 94



(c) threshold = 51



(d) threshold = 43

Figure 16. The binarized images.

method takes 90 msec, a discrepancy that may be attributed to the use of the logarithmic function in the entropic method.

The Between-Class-Variance method was the final choice for the automatic determination binarization threshold for the gradient image produced by the edge detection step. Since the main objective is to classify the pixel based on its gradient as part of the edge or not, it was deemed reasonable to exclude pixels with zero gradient from the threshold determination.

The binarization operation is as follows:

- (1) Obtain the gray level histogram of the image by calling the histogramming routine.
- (2) Determine the zeroth- and the first-order cumulative moments $\omega(k)$ and $\mu(k)$ respectively according to (4.7) for each gray level $k = 0$ to M .
- (3) Let the total mean level μ_T be the $\mu(k)$ at $k = M$.
- (4) For each gray level k , determine the between-class variance σ_B^2 according to (4.16) noting the highest variance value and the corresponding level k .
- (5) Using the gray level k at the maximum σ_B^2 as the threshold, change all pixel gray levels below the threshold to zero (black) and all the other pixels to M (white).

The result is an image with white streaks (edges) in a black background.

EDGE THINNING

Most edge detectors may produce a "thick" edge, since the detectors respond at and near the edge. To improve characterization of the edges, the "thick" edges are thinned by eliminating the "outside" edge one layer at a time until one-pixel thick edges are obtained [20].

A thinning algorithm developed by Zhang and Suen [21] iteratively removes all the contour points that do not belong to the skeleton of the image. So as to preserve the connectivity of the skeleton, each iteration consists of two subiterations.

Consider the positional mapping of a 3×3 neighborhood given in Figure 17, the center representing the position of the contour point P1 considered for deletion or not. For

simplification, let the pixel values be 1 for contour points and 0 for non-contour points. Now, consider a blob or streak of edge pixels that is to be reduced to a skeleton (one-pixel-thick line).

In the first subiteration, a contour point P1 is deleted if it satisfies all of the following conditions:

- (1) it has 2 to 6 neighbors, meaning it is not an end point of the skeleton and that it is on the edge of the blob;
- (2) there is exactly one 01 transition in the ordered set P2, P3, ..., P8, P9 meaning, it is not a part of a line that is already a skeleton;
- (3) $P2 * P4 * P6 = 0$;
- (4) $P4 * P6 * P8 = 0$;

Conditions (3) and (4) mean that the contour point is to be deleted (provided it satisfied conditions (1) and (2), when it is a north-west corner point ($P2 = P8 = 0$), or when it is an east boundary point ($P4 = 0$), or it is a south boundary point ($P6 = 0$).

The second subiteration, still requires conditions (1) and (2) and replaces (3) and (4) by:

- (3') $P2 * P4 * P8 = 0$;
- (4') $P2 * P6 * P8 = 0$;

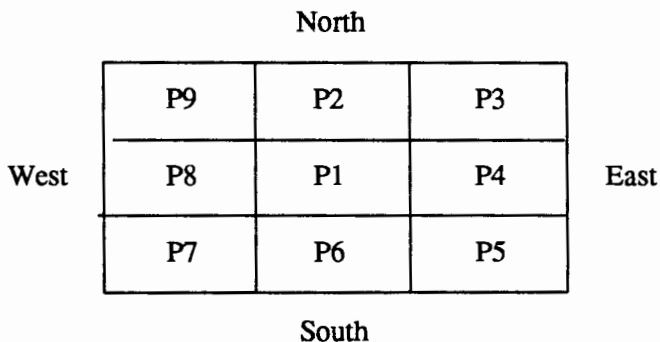


Figure 17. Positional mapping of the 3×3 neighborhood of a contour point P1.

this time deleting the contour point when it is a south-east corner point ($P4 = P6 = 0$), or it is a north boundary point ($P2 = 0$) or it is a west boundary point ($P8 = 0$).

Condition (1) prevents erosion of the line, and condition (2) preserves the connectivity of the skeleton. In one pass, the two subiterations are applied to each edge pixel in the binarized image and the pixel marked for deletion if the conditions are satisfied. After all the pixels have been examined, the marked pixels are deleted. The iterations are repeated until no more pixels may be deleted, resulting in a line-thinned image. Figure 18 shows the effect of the thinning procedure on a binarized image.

The thinning process is an expensive process in the sense that it examines every pixel for exclusion from or inclusion in the final edge line. However, it is an indispensable step if these lines are to be characterized in terms of straight line equations as would be described in the next

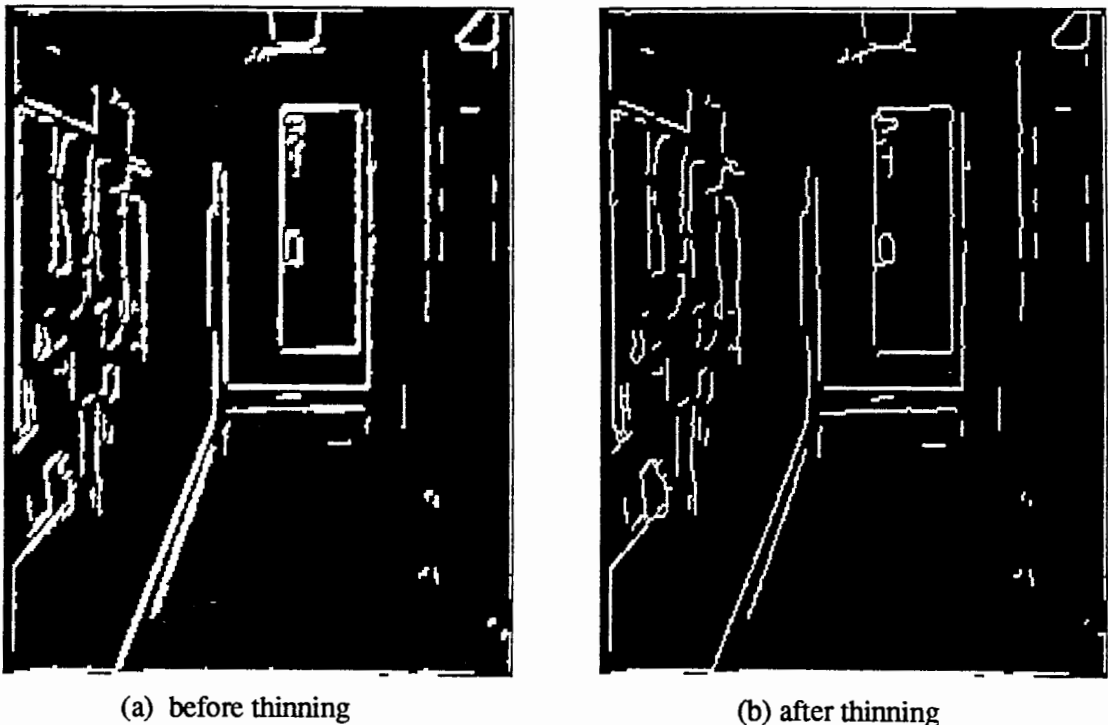


Figure 18. Thinning effect on an image.

chapter, since the characterization process also examines each line pixel and does some parameterization of the line pixels to finally decide on the best description for the edge line. Thus, the main objective of this step is to reduce the computational load and narrow down the possibilities that would be considered as the more probable descriptions.

LABELLING

Considering the amount of attention an edge pixel receives from thinning and subsequent characterization steps, it is desirable to eliminate the pixels that have no chance of being finally taken as line pixels and thus reduce the number of pixels that are to be subjected to further examination and processing. The edge line characterization (as will be described in the next chapter) sets some minimum length to a line segment that will be retained as a valid line. The minimum length in simplest terms is the minimum number of pixels that may lie in the line. It is therefore reasonable to eliminate pixel streaks or blobs whose pixel composition is less than the minimum number for a pixel line so that such streaks or blobs will no longer be examined during the thinning and characterization steps. This "cleaning-up" procedure is implemented by a labelling routine that counts the number of connected pixels comprising a blob or streak and eliminates the blobs that are smaller than a specified pixel area. The eliminated blobs become part of the background. The labelling procedure marks the counted pixels with a gray level other than white or black to keep track of the pixels already examined so that a second binarization step is necessary to convert all the non-black pixels (the surviving blobs, gray level > 0) back to white. The threshold for the binarization this time is simply gray level 1 since all the surviving edge pixels were simply marked non-black. This results in a "cleaner" image, free of very small spots (see Figure 19).

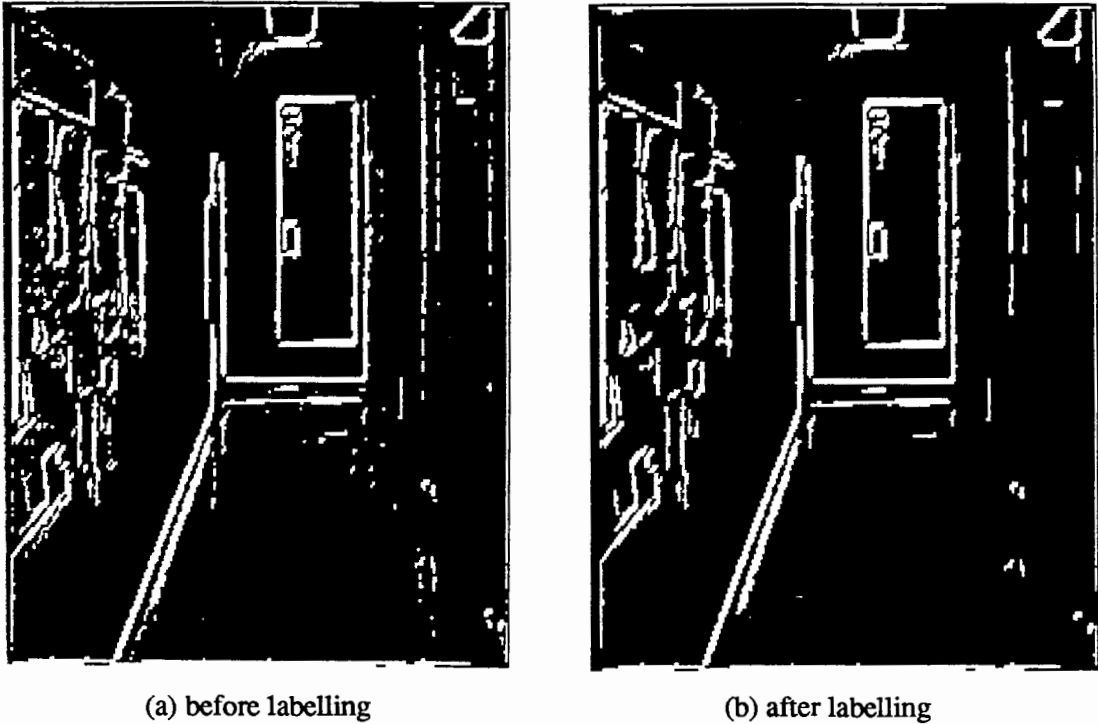


Figure 19. The Effect of the labelling routine.

THE EDGE DETECTION AND EXTRACTION SEQUENCE

Based on the foregoing considerations, the sequence of operations needed to extract the edges and prepare these edges for line feature characterization is as follows:

- (1) edge detection using the Sobel operator;
- (2) gray level histogram determination for the threshold determination;
- (3) threshold determination using the BCV method;
- (4) binarization using the threshold generated by (3);
- (5) labelling using the line length threshold set;
- (6) rebinarization with threshold = 1;
- (7) edge thinning.

CHAPTER V

FEATURE EXTRACTION AND CHARACTERIZATION

The feature extraction method used for a recognition system highly depends on the features of interest and the subsequent characterization amenable to the required recognition task. Image features are closely associated with object boundaries identified in the image. Extraction of such features were addressed in different ways depending on some prior knowledge of the type of objects expected in the image and the intended treatments of the extracted features.

Scenes of the inside of a building are primarily characterized with lines marking wall-floor boundaries, wall corners, posts, door outlines and furniture outlines. The scene may be therefore described in terms of characterizations of line features, e.g., their relative orientations, relative lengths, and relative positions. This chapter centers on the issue of feature extraction and characterization of straight line features.

REVIEW OF LINE FEATURE EXTRACTION TECHNIQUES

Collinearity and proximity of edge points are of course the primary concerns in linear feature extraction. Each researcher has his own way of imposing such constraints. The following works are but a few of the numerous efforts done on the extraction of linear features.

Kahn, Kitchen and Riseman [32] extracted lines by using their so-called connected components algorithm to group pixels with similar intensity gradients into line support regions and fitting lines to these regions. Burns, Hanson and Riseman [33] also grouped pixels of similar gradient orientations into line-support regions and used the structure of the associated intensity surface to determine the location and properties of the line. Nevatia and Babu [34] linked edge elements based on proximity and orientation and approximated the linked elements by piecewise

linear segments. Hung and Kasvand [35] used the chain code and the difference codes on quantized thin lines to identify the "critical pixels" (pixels marking significant bends in the line) and used these critical pixels' positions to determine linear approximations to the lines. The chain code is a sequence of numbers generated by labelling pixels with direction numbers corresponding to a fixed set of orientations on a fixed grid. The sequence of numbers generated by taking the differences of successive chain code elements is the difference code which indicates the relative direction of the chain code segments. Shneier [36] made use of the pyramid structure in his line extraction process. His method involves building a series of successively lower resolution images from the original image, applying line-detector masks to each level followed by a line enhancement step and grouping the line-response points into line segments by means of a stepwise clustering process. His stepwise clustering process first groups points with similar direction and then subdivides each group on the basis of the separation between the points. The number of points in each subgroup determines the existence of the line. The same treatments are used for each pyramid level, therefore, each line in the lower resolution level corresponds to elongated lines in the original image. The extraction and representation allows for finding relevant areas in the image for further examination or processing.

An interesting approach to line feature extraction is the work by Princen, Illingworth and Kittler [37]. They presented a hierarchical approach to line extraction based on the Hough Transform. Line segments are identified in small subimages using the conventional Hough Transform parameterization and these short line segments were grouped into longer ones at each higher level of the hierarchy. They used the overlapped pyramid structure for the hierarchical grouping. The pyramid structure was also used by Shneier [36, 38], Rosenfeld [39] and Hong [40] in edge feature extraction but employing much different techniques for extracting the lines and relating the higher level lines with their lower level components.

Of the linear feature extraction techniques, the Hough Transform method seems to be the most tolerant to missing edge points and random extraneous data which are almost always

inherent in digitized real images. The Hough Transform method is also applicable to non-linear shapes provided the shape can be described in terms of some parametric curves [2, 41]. Illingworth and Kittler [42] made a comprehensive review of the Hough Transform method and the research done in the area. The review cited several desirable features of the method that make it superior to other boundary-based feature extraction techniques for shape and even motion analysis in natural images. Most natural images contain noisy, missing and extraneous data. Among the advantages of the Hough Transform method are: (1) the method treats each edge point independently, making implementation in more than one processing unit possible; (2) it combines events based on the transform space rather than the input image thereby making it tolerant of partial or slightly deformed shapes in the image; (3) it is very robust to the addition of random data produced by poor image segmentation; (4) it can simultaneously accumulate evidence for several occurrences of a particular shape in the image.

However, the standard implementation of the Hough Transform method entails large storage and computational requirements. The review also described some work done to overcome this drawback like the use of small-sized accumulator arrays and the use of extra data to restrict the range of parameters which need to be addressed in the case of non-linear shapes.

THE HOUGH TRANSFORM METHOD

The problem of line or curve detection in general involves establishing meaningful groups of edge points that lie along a line or curve. The Hough Transform method is a classical way of detecting edge points that satisfy the collinearity constraint for straight lines [37] without the strict pixel-connectivity restriction imposed on the edge points. The Hough Transformation involves the mapping of points in image space to sets of points in an appropriately quantized parameter space based on some parameterization scheme of the shape model expected in the image.

Straight Line Parameterization

A set of collinear points in image space may be described by the slope-intercept form for the equation of an infinite length straight line:

$$y = m x + c \quad (5.1)$$

where (x, y) is a point on the line in image space and (m, c) are two parameters corresponding to the slope and the y -intercept of the line, respectively.

Each straight line has its own unique (m, c) pair, so that when mapped to the m - c parameter space, a line will be a single point in the parameter space. When a point (x, y) is sampled for all possible (m, c) pairs in the parameter space which satisfies equation (5.1), a straight line in the parameter space results as described by the transposed form of equation (5.1), i.e.,

$$c = -x m + y$$

where (m, c) is a point in the parameter space line and $(-x, y)$ are the slope and the c -intercept of the line in the parameter space, respectively.

If each point (x_i, y_i) maps to a straight line in the m - c parameter space, then the point of intersection of these parameter space lines defines the unique (m, c) parameters for the line in the image space. Figure 20(b) shows the m - c parameterization of 3 points in the straight line of Figure 20(a).

However, the above straight line parameterization form fails for vertical and near vertical lines when $m \rightarrow \infty$. A more robust form is the parameterization based on the normal form of the straight line equation:

$$\rho = x \cos\theta + y \sin\theta \quad (5.2)$$

where ρ is the length of the normal vector and θ is the angle the normal vector makes with the x -axis.

With the same principle as the slope-intercept parameterization, each point (x_i, y_i) therefore, maps to a sinusoidal curve in the ρ - θ parameter space. When θ is restricted to the interval

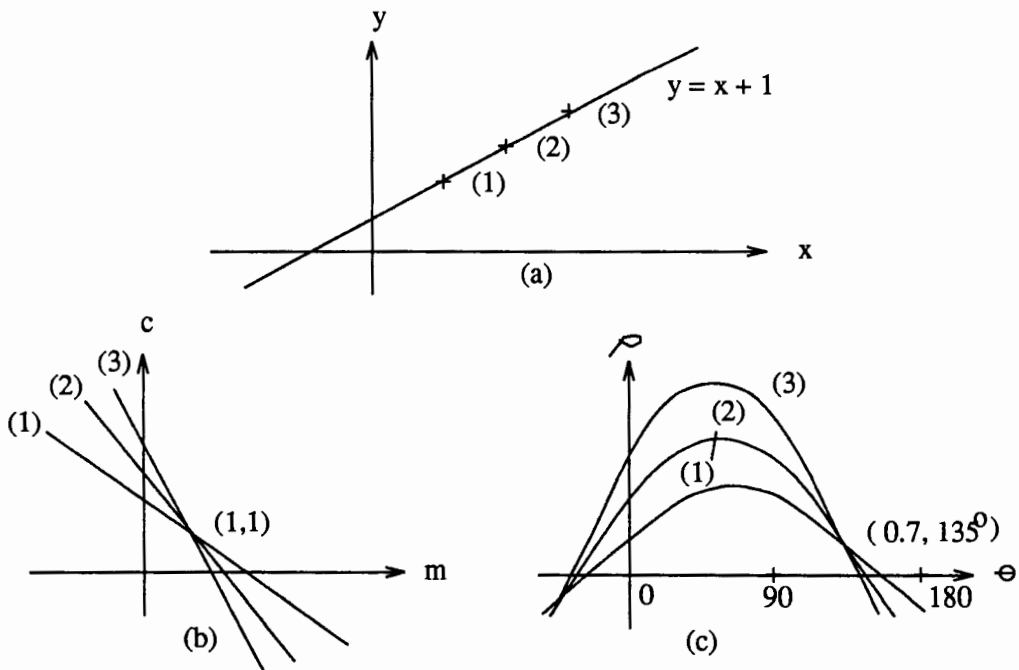


Figure 20. Straight line parameterizations. (a) the straight line, (b) slope-intercept parameterization, (c) normal parameterization.

$(0, \pi)$, the parameter space sinusoids that describe all the points in the image space line will intersect at a unique point (ρ, θ) in the parameter space. Thus, an intersection point in parameter space defines the unique (ρ, θ) parameters of the image space line formed by the points. The ρ - θ parameterization of the line in Figure 20(a) is shown in Figure 20(c).

Hough Transform method of Line Extraction

The key idea of the Hough Transform method as applied to straight line extraction is the mapping of image points to the parameter space based on the normal form of the straight line equation and identifying the straight lines based on events in the parameter space. The straight line equation in the x - y image space is viewed as a mutual constraint between image space points and parameter space points. This treatment defines a one-to-many mapping from an image point to a set of parameter values. Similarly, a point in the parameter space maps to all the

points in the image space that satisfy the parameter value pair represented by the parameter space point. Thus the number of parameter space curves that intersect at a parameter space point gives the number of points in the image space that comprise the line described by the parameter value pair.

The standard implementation of the Hough Transform method for straight line extraction involves the use of a two-dimensional accumulator array to represent the quantized parameter space. The parameterization involves the sampling of each edge point in the image space with each θ value in the parameter space using equation (5.2) and incrementing the corresponding (ρ, θ) cell in the accumulator array. After all the edge points have been parameterized, the accumulator array is then scanned and cells with counts higher than a preset threshold are taken as straight line indicators. The count is equivalent to the number of collinear edge points and the parameter values of the line are obtainable from the coordinates of the cell.

The fineness of the quantization must be so chosen as to reflect the accuracy of the determination of the parameters. In a study of the discretization errors in the Hough Transformation, Van Veen and Groen [43] suggested that the sampling is optimal when:

$$\Delta\rho = l \sin \frac{\Delta\theta}{2} \quad (5.3)$$

where l is the length of the segment in the image space, $\Delta\rho$ and $\Delta\theta$ are the quantization intervals for the parameter space.

APPROACHES TO THE USE OF THE HOUGH TRANSFORM METHOD

The capability of the Hough Transform to simultaneously extract straight lines in an image space plus its high tolerance to missing edge points or extraneous edge points makes it an appropriate method for extracting lines from images of natural scenes.

A Global Application of the Hough Transform Method

The simplest implementation of the method is by the parameterization of all the edge points in the whole image into a single accumulator array and scanning the array for counts greater than a threshold length set for a valid line. However, this requires a huge accumulator array if an optimal quantization interval is to be observed. The simplest way to implement the accumulator is with the use of an array-type data structure, setting aside as much computer memory data space as required by the parameterization space. Implementation on a PC might pose some problems because of the way memory space is segmented according to the operating system used. Assuming that memory data space is limitless, still, allocating such huge space is not wise engineering because only a number of the array cells will indicate the valid lines and some cells may not even accumulate any counts, meaning a big chunk of the space is not really useful in the final analysis. A hash table or a linked list implementation may be more economical but these implementations may also present drawbacks in the ease and speed by which the cells may be accessed.

The transform's independent treatment of edge points may also be viewed as a weakness in the global application because it could result in accidental associations of edge points [44] which by accident are collinear but really belong to some other true line. Accidental associations may occur for example when there are several true lines that may all be intersected by one line that does not really exist. All the intersection points are of course collinear and would vote for the same cell in the accumulator array and if the number of points satisfy the threshold set for valid lines then the false line will be declared to exist. Also, line segments that happen to be collinear will be declared as one long line regardless of their spatial separation.

Finally, localization of the lines in the image space must be considered. Since the parameter values describe an infinite length line, finding the specific location and extent of the line in the image space may require some back-transformation to find the endpoints of the line.

The Hierarchical Approach

Splitting the whole image into smaller subimages and implementing the transformation on the smaller subimages means a smaller accumulator array [42] plus localization of the lines detected to the subimage region [37]. Localizing the parameter space to each subimage reduces the possibilities of accidental associations of points into a line and eliminates the combination of widely separated collinear segments into single lines. The location of the subimage region also provides the exact location of the lines in the image. This application of the Hough Transform method results to a number of short lines, each line localized in its subimage region.

Since the ultimate goal is to come up with a global description of the image in terms of the lines found in the image, it is then a matter of grouping the line segments into longer lines according to some collinearity and proximity constraints. To maintain the proximity requirement, a neighborhood of subimages is treated as a single subimage, and line segments detected in these subimages that happen to be collinear are grouped into longer lines. This results to a set of longer lines each one localized to its bigger subimage region.

The process of grouping lines in neighboring subimages to form longer lines in the combined neighborhood is continued until no more neighboring subimages may be combined into larger areas or no more lines can be grouped together. Each grouping of lines in a neighborhood of subimages into longer lines in the bigger subimage constitutes one level of the hierarchy. Thus, the lowest level of the hierarchy consists of the image containing the edge pixels, the first level consists of the line segments in each subimage of the input image, the second level consists of grouped first level line segments in neighboring subimages, and so forth.

THE LINE EXTRACTION SCHEME

The hierarchical approach is adopted for the line extraction problem in this application considering the solution it offers to the problems of accumulator array size, accidental associations of points and line localization.

The Hierarchical Structure

The hierarchical approach uses the concept of the overlapped pyramid structure. Each level of the pyramid consists of lines found in the subimage components of the level. This representation makes it different from the conventional pyramid structure which consists of successively lower resolution images [2, 38]. The pyramid concept was used in the way the subimages are grouped into bigger subimages to reflect the extent of collinear line segments in successive levels of the hierarchy. At each level, a neighborhood of 2×2 subimages in the lower level is treated as one subimage as shown in Figure 21.

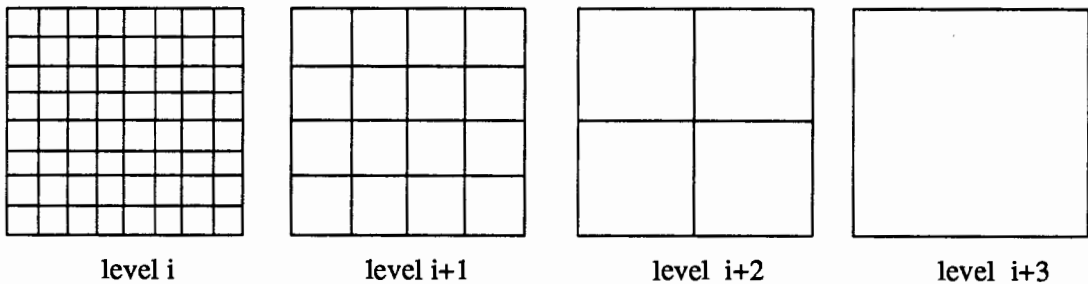


Figure 21. Grouping of 2×2 subimage neighborhoods into bigger subimages.

The lowest level of the hierarchy consists of short line segments determined from subimages that comprise the binary line-thinned image resulting from the edge extraction operations. The pyramid is built from the bottom to the top by grouping line segments from 2×2 adjacent subimages (to be called central subimages) to form longer line segments in the next higher level, provided there is sufficient support for the existence of the line from within the central subimage region or from the immediate neighbors of the central subimage region. The subimages in the 4×4 neighborhood which is composed of the central subimages and their immediate neighbors will be called sibling subimages and the region corresponding to the span of the central subimages in the higher level will be referred to as the parent subimage. Figure 22 shows a neighborhood of sibling subimages used to form longer line segments for a parent subimage.

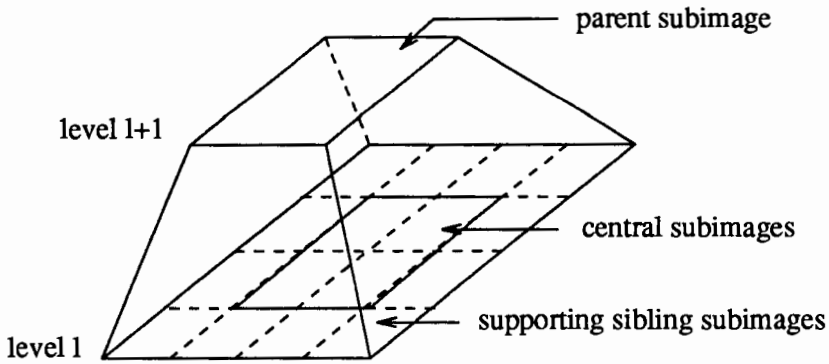


Figure 22. Neighborhood of subimages used to form line segment groups for a parent subimage.

It must be noted that the parent subimages define the subimage components of the new level. The central 2×2 regions that make up the parent subimages are disjoint. However, they share rows and columns with adjacent central subimage regions to provide support information for the confirmation of the line existence in the parent subimage. Figure 23 shows the overlapped regions for adjacent subimages. This technique was first proposed by Shneier [38] for his edge pyramid and was later used by Princen, Illingworth and Kittler [37] for line segment grouping.

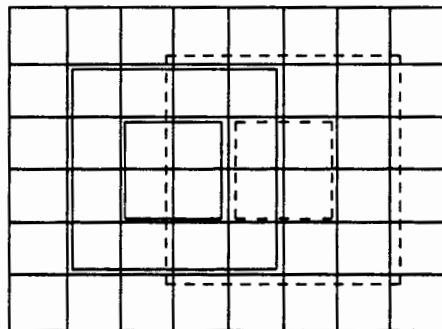


Figure 23. Overlapping regions for adjacent central subimages.

Formation of new levels continues as long as new parent subimages may be formed from the lower level sibling subimages. The formation of the parent subimage is actually implemented in terms of the grouping of collinear line segments in the lower level sibling subimages. This results in a longer line segment that is indicated to exist inside the parent subimage.

The Data Structure Representation of the Hierarchy

The hierarchical structure is implemented as a hierarchy of data structures as shown in Figure 24. The pyramid itself may be viewed as a list of levels (Figure 24(a)) with the head of the list corresponding to the highest level. Considering the manner by which the line segments are grouped, the most obvious representation of each level is by an array of cells, each cell

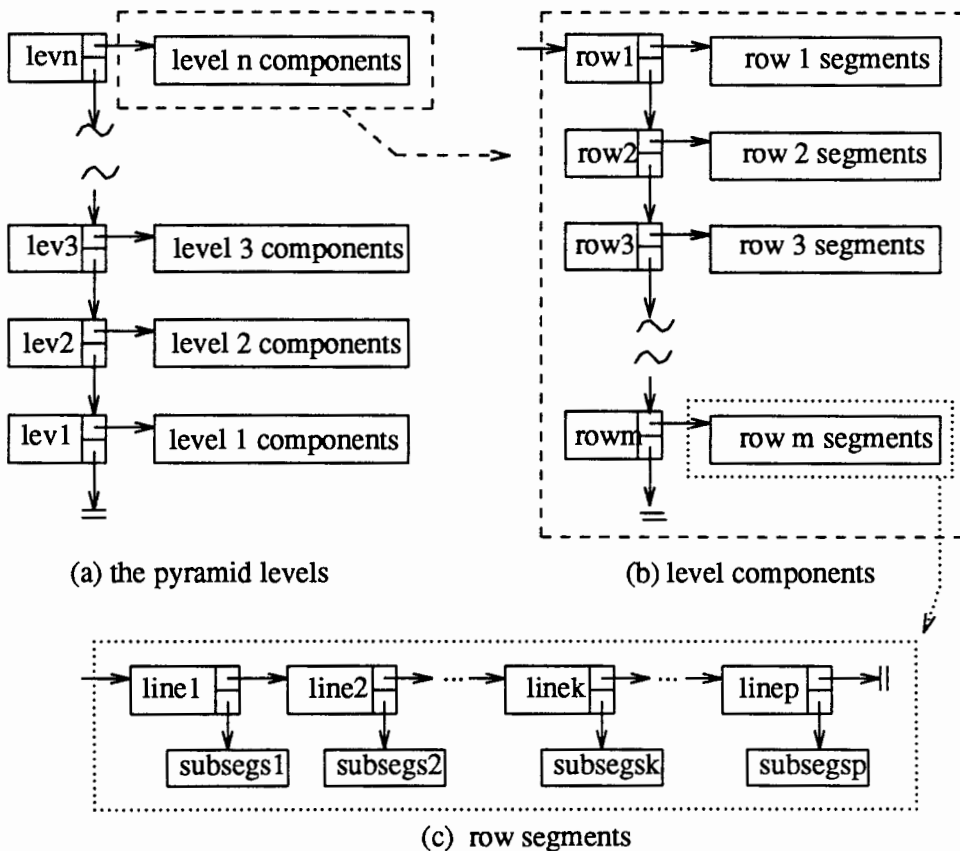


Figure 24. Hierarchical representation of the hierarchical structure.

containing the line segment groups that exist in each subimage component of the level. However, because only the longer line segments propagate to higher levels, it is highly probable that not all the subimage components will contain line groups. It was then decided to represent only those subimage components that contain line groups. To preserve the topology of the subimage partitioning for the level, the lines are listed according to the order by which their subimage locations occur in the image. This was effected by representing each level as a list of rowlists (Figure 24(b)). Each rowlist corresponds to a row of subimages of the partitioning. The rowlist contains all the line segments found along the row (Figure 24(c)). Each line segment in turn is attributed with a pointer to the local center of its subimage location, its normal parameters, and a list of the lowest level subimage regions that contain the subsegments comprising the line.

The Lowest Level Line Segment Determination

At the lowest level, the binary line-thinned image is partitioned into subimages of size L ($L \times L$ pixel neighborhoods). The line segment determination in each subimage involves the ρ - θ parameterization of each edge point, incrementing the appropriate accumulator cell for each parameter point found and finally scanning the accumulator cells for counts that satisfy the threshold count for valid lines.

As shown in Figure 25, the expanse of the L -sized central subimage implies that only lines intersected by the corresponding normal vector inside the circular region may exist inside the central subimage. This sets the parameter space limits for the valid lines to be:

$$-\frac{L}{2} \leq \rho \leq \frac{L}{2}, \quad 0 \leq \theta < \pi \quad (5.4)$$

However, the shortest possible line that may pass the edge of the circular region will not be detectable from the accumulator counts even if it is a part of a longer line that happens to pass through the subimage. In order to make these lines detectable, it is necessary that the immediate neighborhood of the subimage be considered to provide support information for the line. Thus, an overlapped region of size $2L$ that contains the subimage as the central part and $L/2$ rows and

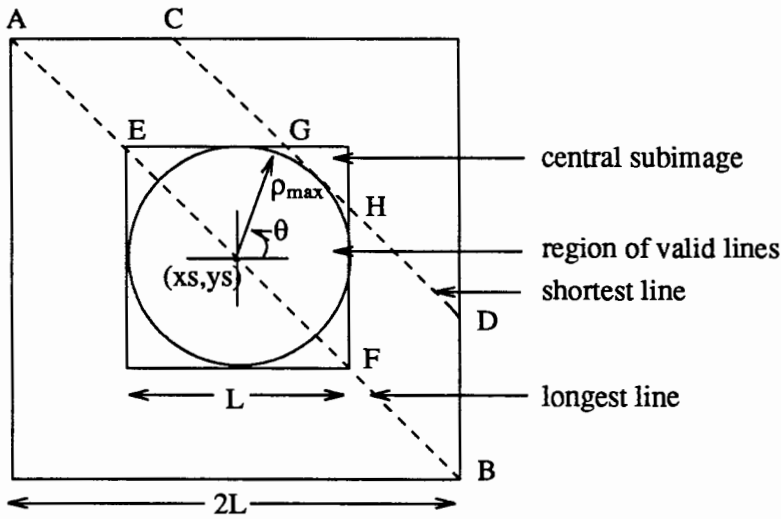


Figure 25. An overlapped subimage for the lowest level line segment determination.

columns of pixels that immediately surround the subimage was considered for the Hough Transformation. The use of the overlap also contributes to a more uniform distribution of detectable counts. This is because the ratio of the lengths of the longest and the shortest lines, $\overline{AB}/\overline{CD}$ is 1.55 for the scheme with overlaps, which is much lower than the ratio $\overline{EF}/\overline{GH} = 3.41$ for the scheme with no overlaps. This uniformity helps in setting a safe threshold value for the counts that will be taken as indicative of valid lines, i.e., taking a reasonable fraction of \overline{AB} as the threshold will be sufficient to detect enough counts for CD as well. Whereas, for GH to be detectable, the threshold must not be greater than the length \overline{GH} , which however, is too small for EF . Thus, with the overlap, imposing that 50% of the line must be within the region to be detectable is sufficient to detect the properly supported short segment, whereas, a much lower requirement must be set for the non-overlapped case. Setting a low pixel count as a threshold will result in indications of short segments that are really parts of longer segments, thereby introducing redundant results.

The size of the region and the parameter space limits dictate the quantization intervals $(\rho_{\Delta}, \theta_{\Delta})$ which must be sufficiently small to distinguish all possible valid lines in the region. The

number of possible line orientations in the overlapped region is taken as $4L$. This requires the parameter space quantization intervals to be:

$$\theta_{\Delta} = \frac{\pi}{4L}, \quad \rho_{\Delta} = L \sin \frac{\theta_{\Delta}}{2} \quad (5.5)$$

Thus, the accumulator array needed to represent the parameter space will consist of $(\frac{L}{\rho_{\Delta}} + 1) \times 4L$ cells.

It must be recalled that the normal parameters (ρ, θ) are referenced to the center of the x - y coordinate system. In a similar sense, the (ρ, θ) parameters found for the collinear edge points in the subimage are referenced to the local center of the subimage. For the benefit of notational convenience, let the local parameters of the the line segment be denoted as (ρ_o, θ_o) , the local coordinates of the edge points as (x_l, y_l) and the local subimage center as (x_s, y_s) . Thus, the detected line segments may be thought of as collinear edge points that satisfy equation (5.2), i.e.,

$$\rho_o = x_l \cos \theta_o + y_l \sin \theta_o$$

Higher Level Grouping

Line segments from neighborhoods of 4×4 subimages containing the 2×2 central subimages are grouped into longer line segments, provided that the line segments are collinear in a sense that will be explained shortly. Each line segment detected in a lower level may be described as a feature point located at the intersection of the line and its normal in the lower level subimage region, i.e.,

$$x_o = \rho_o \cos \theta_o, \quad y_o = \rho_o \sin \theta_o.$$

Figure 26 shows a neighborhood of subimages containing a straight line represented as feature points marked at the foot of the normal (x_o, y_o) , local to each subimage involved.

Since each feature point is represented with respect to its local origin (x_s, y_s) and must now be considered with respect to the center of the parent subimage (x_p, y_p) , the appropriate

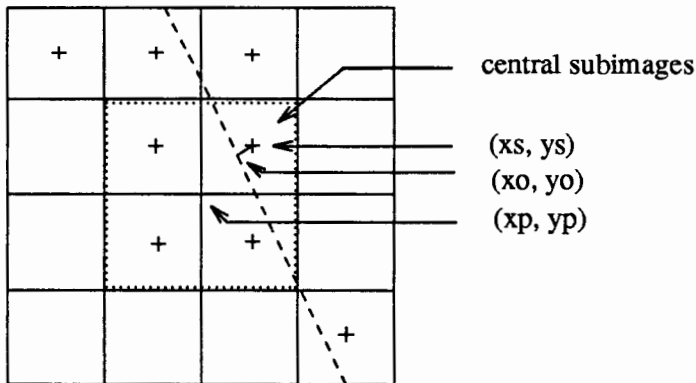


Figure 26. A neighborhood of subimages containing collinear line segments.

parameter space curve is determined by adjusting the x and y of equation (5.2) for the local centers of the subimages as:

$$\rho = (x_o - (x_p - x_s)) \cos\theta + (y_o - (y_p - y_s)) \sin\theta. \quad (5.6)$$

The conventional Hough Transform parameterization scheme is used to find the collinear line segments in the neighborhood of feature the points using equation (5.6) for the ρ - θ mapping of the feature points. However, the parameterization introduces possible discretization errors $\Delta\rho$ and $\Delta\theta$ (different from the sampling intervals ρ_Δ and θ_Δ). $\Delta\rho$ is taken to be $\sqrt{2}$, the minimum bar width that can include all edge points for a line of any angle if the line is drawn on a discrete grid with a point spacing of one pixel. $\Delta\theta$ is the θ sampling interval used in the lower level. The possible ρ , θ values for a feature point (x_o, y_o) in the new parameter space will therefore be given by:

$$\theta_o - \frac{\Delta\theta}{2} \leq \theta \leq \theta_o + \frac{\Delta\theta}{2}, \quad (5.7a)$$

$$\rho_o - \frac{\Delta\rho}{2} \leq \rho \leq \rho_o + \frac{\Delta\rho}{2}. \quad (5.7b)$$

These set the collinearity constraints for the feature point (x_o, y_o) , meaning that any other feature point that would have a (ρ, θ) value within this range is collinear with the point (x_o, y_o) .

The bounds given by the relations (5.7a) and (5.7b) narrow down the range of the parameter space that need to be examined for collinear feature points.

Just like the lowest level scheme, the feature points in the sibling subimages are parameterized, this time according to equation (5.6) and only within the range of the collinearity constraints for the feature point. The appropriate accumulator cell is tabbed for each parameter point found. Similarly, a longer line segment is detected if there are collinear line segments in at least two neighboring subimages of the 4×4 neighborhood. Line segments identified to be collinear are grouped together and treated as a single longer line segment. The longer line segment is attributed now with its local parent subimage center represented as its (x_s, y_s) , its local parameters (ρ_o, θ_o) and a list of member subsegments. Each member subsegment is represented as a pointer to the center of the lowest level subimage region that contains the subsegment. The segments that become members of a new group at the higher level are deleted from the lower level and their lists of member subsegments are now linked to the new group's subsegment list. The grouping scheme terminates shorter lines at lower levels in the hierarchy and allows only longer lines to participate at higher level groupings, thereby reducing the amount of calculations needed at higher levels.

The quantization intervals are set in the same manner as was used in the lowest level according to (5.5) where the L is now the size of the central region which is twice the size in the preceding level. The assumed error bound $\Delta\theta$ given by the relation (5.7b) and the sample spacing θ_Δ imply that only five samples of θ are relevant for each feature point. The width $\Delta\rho$ from relation (5.7a) and the sample spacing ρ_Δ imply that the number of cells that need to be incremented along the ρ axis for each θ value is equal to the rounded off value of $\frac{\sqrt{2}}{\rho_\Delta} + 1$. The sparsity of the feature points in the sibling subimages also suggests that the conventional accumulator array (which indeed grows larger as the level in the hierarchy increases), may be represented as a linked list with elements only for the relevant cells.

THE IMPLEMENTATION OF THE HIERARCHICAL HOUGH TRANSFORM SCHEME

The hierarchical scheme for the line feature extraction and characterization using the Hough Transform technique is divided into 2 routines:

- (1) the lowest level line segment determination from the binary line-thinned image;
- (2) the high level line segment grouping.

To simplify references to the routines, let *hhlow* refer to the lowest level line segment determination routine, *hhhigh* refer to the high level line segment grouping routine, and *hhough* refer to the main program that manages the calls to *hhlow* and *hhhigh*. Let *exp 2*-square refer to a square of $2^n \times 2^n$ pixels.

The first step in the preparation of the input image for the hierarchical treatment is the determination of the smallest *exp 2*-square that may encompass the image. The image is then centered in this square to allow a uniform partitioning of the image into subimage regions at each level of the hierarchy. The main algorithm implemented may be summarized as follows:

- (1) center input image in smallest *exp 2*-square;
- (2) call *hhlow* to generate the first level, given the *exp 2*-square-centered input image;
- (3) call *hhhigh* to generate the second level, given the first level;
- (4) as long as a new level is generated, repeat calling *hhhigh*, each time giving it the last level generated.

The C program source codes written for the implementation are given in Appendices D, E, F, G and H. Appendix D is a listing the `#include` file for the main program and the subprograms listed in Appendices E, F, G and H. It contains the data structure definitions for the hierarchical representation used and all the other definitions pertinent to the implementation. Appendix E contains the main Hierarchical Hough Transform Program *hhough* that calls *hhlow* and *hhhigh* accordingly and manages the hierarchical levels and the routines that are common to *hhlow* and

hhigh. Appendix F contains *hhlw* and its pertinent subroutines. Appendix G contains *hhigh* and its pertinent subroutines. Appendix H contains the routines used for generating a print out of the line segment descriptions and for remapping the lines found in each level to the image memory for display on the external monitor. Only the routines in Appendices E, F and G are necessary for the line feature extraction task for real-time application. The routines in Appendix H are useful for monitoring the development of the hierarchical structure.

The Lowest Level Line Segment Determination Implementation

The lowest level line segment determination scheme described is implemented according to the following pseudocode:

```

/* lowest level line segment determination */
/* begin hhlw */
{ Do initialization steps
  while (an image row is part of subimage row)
  { Set up the subimage row in buffer
    Set up first subimage
    while (an image column is a part of the central subimage)
    { List edge points found in overlapped subimage
      if ((edge points count >= threshold) && (central points count > 0))
      { Parameterize edge points
        FindPeaks in the accumulator array
        if (linesegments found)
          link up linesegments into linelist
      }
      Set up next subimage
    }
    if (linelist is not empty)
    { attach rowmarker to head of linelist
      if (is first rowlist)
        mark as head of rowlists
      else
        link up to rowmarker of last rowlist
    }
    Shift out upper L rows in buffer, shift up lower L rows
  }
  return (list of rowlists)
}
/* end of hhlw */

```

The initialization steps consist of setting up buffers to hold the pixel rows that comprise a row of overlapped subimages, setting up space for the accumulator array, setting up a sine-cosine look-up table for use in the parameterization, setting up pointers to the first subimage row in the image row buffers and setting up the Hough Transform parameter space settings. The image rows that comprise the overlapped subimages are buffered so as to avoid frequent access to the image memory during the processing of the subimages. It was considered reasonable to set up a sine-cosine look-up table for the parameterization so as to minimize calls to the sine and cosine functions in the compiler's math library. It must be noted that each edge point will be sampled for all possible θ in the parameterization range. The accumulator array is set up only once, and since the line segment determination is done for each subimage separately, the same accumulator array is used for each subimage.

Subimage settings. The subimage size used for the lowest level line segment determination is $L = 4$. This partitions the $exp2$ -square-centered binary line-thinned input image into subimages of 4×4 pixel neighborhoods. The overlapped region for a subimage is the 8×8 pixel neighborhood with the 4×4 pixel subimage centered in this neighborhood.

Hough Transform parameter space settings. Based on the considerations described earlier, the parameter space sampling intervals are as follows:

$$\theta_{\Delta} = \frac{\pi}{4L} = \frac{\pi}{16}$$

$$\rho_{\Delta} = L \times \sin \frac{\theta_{\Delta}}{2} = 0.40$$

Therefore the quantized parameter space settings are:

	number of intervals	range
theta axis:	$4L = 16$	$[0, \pi]$
rho axis:	$\frac{L}{\rho_{\Delta}} + 1 = 11$	$[-2, +2]$

An 11×16 accumulator array was used to represent the quantized parameter space, each cell corresponding to a point in the quantized space. Since array cell indices are always positive

integers, the ρ -axis must to be shifted so that all the ρ -ordinates may be addressed in terms of the row indices. This requires that each of the ρ values computed in terms of the quantized space (ρ_q) will have to be adjusted by:

$$\rho_q' = \rho_q + \rho_{zero}$$

where ρ_{zero} is the index for the middle row that corresponds to the center of the quantized ρ -axis ($\rho_q = 0$).

The threshold for the counts that indicate a valid line is set at 4, assuming that at least 50% of the points on the line must exist in the overlapped image to indicate the line as valid.

Line segment determination in the subimage. Prior to the parameterization, the number of edge points in the overlapped subimage is first counted. If the number of edge points is at least equal to the threshold set and there is at least one edge point in the central subimage, then the subimage is subjected to all the rigors of the Hough Transform method, otherwise the subimage is simply discarded. As was stressed in earlier discussions, each edge point is sampled for each θ in the quantized space. The appropriate accumulator cell is incremented for each (ρ, θ) determined, and the accumulator is finally scanned for counts that satisfy the threshold. Each identified line segment is represented as a list element containing its local (ρ_o, θ_o) and the local center of its subimage region (x_s, y_s) , which is then linked to the list of segments found in its subimage row.

Level 1 of the hierarchy is therefore composed of at most 60 lists of line segments for the 256×240 image, each list corresponding to each row of subimages.

Higher Level Line Segment Grouping Scheme Implementation

The grouping of collinear line segments in a neighborhood of lower level subimages is very similar to the scheme used for the low level line segment determination. This time the elements of the subimage are no longer image pixels but feature points representing the lower level lines. A 4×4 array of pointers to line segment elements in the lower level rowlists now keep

track of component feature points in each 4×4 sibling subimages that comprise the overlapped subimage and the inner 2×2 pointers keep track of the feature points in the central subimages. Given the list of rowlists that constitutes the last level, *hhigh* returns a list of rowlists for the new level formed. The pseudocode for the implementation of the high level grouping scheme is as follows:

```

/* higher level line segment grouping */
/* begin hhigh */
{ while (lowlevel rowlist is part of central subimage row)
  { Set up pointers to lowlevel rowlists comprising the subimage row
    while (not end of all the lowlevel rowlists of row)
      { Set up the siblings window
        if ((feature points  $\geq$  threshold) && (central point  $>$  0))
          { Label central points (line segment elements in central siblings)
            for (each feature point in the central siblings)
              { Get collinearity votes from immediate sibling points
                Find a group from the highest-voted ( $\rho$ ,  $\theta$ ) cell
                if (group found)
                  { mark feature points as "grouped"
                    link up new line segment into linelist
                  }
                else
                  mark the feature point as "not grouped"
              }
            Finalize Labels of central points
          }
        Shuffle pointers to the next siblings window
      }
    if (linelist not empty)
      { get rowmarker for linelist
        if (first rowlist)
          mark as head of rowlists
        else
          link up new rowlist to head of last rowlist
      }
    Delete grouped segments from the upper 2 lowlevel rowlists
    Shove out the upper 2 lowlevel rowlists, include the next 2 rowlists
  }
  Delete grouped segments in the last group of lowlevel rowlists
  if (list of rowlists not empty)
    Delete rowmarkers to empty lowlevel rowlists
  return (list of new rowlists)
}
/* end of hhigh */

```


The subimage size at each new level is the combined size of the 2×2 neighborhood of central sibling images. Thus, the subimage size L for a new level is simply twice the size in the last level.

The quantized parameter space settings are determined in the same manner as that in the lowest level. Table II presents a summary of the quantized parameter space settings for each of the levels for a 256×240 input image centered in a 256×256 square.

TABLE II
SETTINGS FOR THE HOUGH TRANSFORM QUANTIZED PARAMETER SPACE

level	subimage size L	sampling intervals		no. of intervals		ρ range $-L, +L$	ρ_{zero}
		θ_{Δ}	ρ_{Δ}	θ	ρ		
1	4	$\frac{\pi}{16}$	0.4	16	11	-2, +2	5
2	8	$\frac{\pi}{32}$	0.4	32	21	-4, +4	11
3	16	$\frac{\pi}{64}$	0.4	64	41	-8, +8	21
4	32	$\frac{\pi}{128}$	0.4	128	81	-16, +16	41
5	64	$\frac{\pi}{256}$	0.4	256	161	-32, +32	81
6	128	$\frac{\pi}{512}$	0.4	512	321	-64, +64	161
7	256	$\frac{\pi}{1024}$	0.4	1024	641	-128, +128	321

The tabulation shows that as the level increases, the size of the accumulator array also increases. However, as explained earlier, only the θ values within $\pm \frac{1}{2} \Delta \theta$ and the ρ values within $\pm \frac{1}{2} \Delta \rho$ need to be sampled for each feature point. Since $\Delta \theta$ is one half of the θ_{Δ} of the previous level and $\Delta \rho$ is constant at 0.707, then only 5 samples need to be taken for the θ dimension and 3 samples for the ρ dimension. Thus, only the accumulator array cells required for the sampling were considered and represented as a list of 5 sets of ρ cells. Each set of ρ cells represents the 3 consecutive cells in the conventional array that lie within the range of the ρ values that need to be sampled. Each cell consists of a counter field for the accumulation and a field where the identity of contributors to the count are noted. This provides a way of tracking the feature points that

belong to a group if ever one is indicated. The same space was used for the groupings in all the levels, each time initializing the space by assigning the appropriate accumulator array coordinates to the cells according to the range established by the feature point's collinearity constraint. For simplicity, let this subspace of the parameter space be called the voting array.

Similar to the low level scheme, there must be enough feature points in the sibling subimages and there must be at least one in the central region for the line segments to be considered for grouping. Each feature point establishes the range of ρ and θ values that need to be used for the sampling. For each central feature point, each point in the immediate neighboring sibling subimages is sampled, provided the ranges of θ for the central feature point and the neighboring point overlap. The count in the appropriate cell of the voting array is incremented by 1 for every (ρ, θ) pair that is within the range of the voting array. The "identity" of the voter is also noted in the cell if it is a central feature point. A group corresponding to a longer line segment is formed when a cell indicates more than one vote. Since a feature point can be assigned to 15 possible lines according to its collinearity constraints, and each neighboring feature point may vote for these possibilities as its own collinearity constraints will allow, it is then always possible that several cells may accumulate more than one vote. Thus, to get the best line description of a new group, the voting array is searched for the cell that obtains the highest vote. The parameter space point represented by the cell is then taken as the set of (ρ, θ) parameters for the new group. Since we are now dealing with short line segments that are being combined into longer line segments, a feature point is allowed to be a part of only one group. To ensure this, all the central feature points that get included in a group are appropriately marked as "grouped" feature points, so that they will no longer be considered in the formation of other groups within the same parent subimage.

Each feature point must be initially labeled to be able to keep track of the identity of the central feature points. This also differentiates the central feature points from those of the supporting sibling subimages. It must be recalled that the feature points in the sibling subimages

outside the central subimage region simply provide support information for the existence of a group but do not get included in the group formed for the parent subimage. The label assigned to a central feature point is updated during the grouping process depending on whether the feature point becomes a group member or not. The labels are finally changed after all the central feature points were considered to mark the feature point for deletion from its lower level if it became a group member in the higher level or for retention in its level if its presence was not indicated in the new level.

When a line segment becomes a member of a new group, its list of subsegments is detached from the element used to represent the line segment and linked to the new group's subsegment list. After a new rowlist is completed, the last level's rowlists used for the grouping are cleared of the elements that were indicated as "grouped" and have no more subsegment list anyway. These elements are detected by the finalized labels. Using the labels as the basis for deletion rather than the subsegment list allows the use of the same cleaning up treatment for all levels including the cleaning up of the first level lines which do not have subsegment lists. Finally, when the whole level is completed, the last level is cleared of empty rowlists, thereby cleaning up the remnants of the segments that have propagated to longer lines and leaving only those lines that really belong to the level.

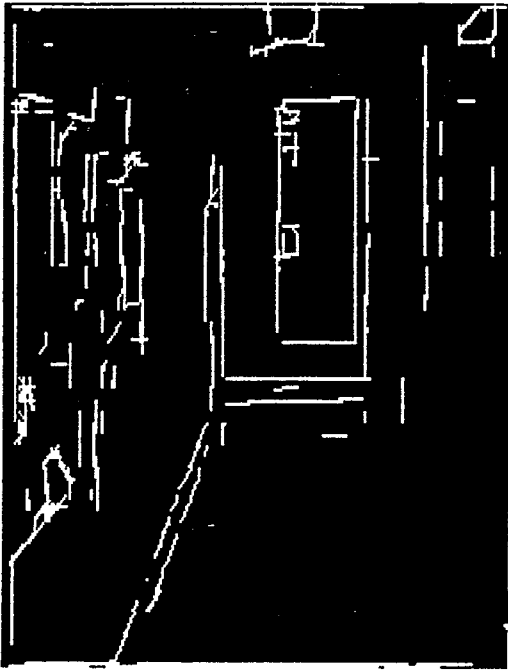
The images in Figures 28 and 29 show the lines formed at each level when the aforementioned scheme was applied to the binary line-thinned image shown in Figure 27. The first level remap (Figure 28(a)) shows that isolated streaks shorter than 4 pixels were not detected which is of course expected. It is also noticeable that small discontinuities and/or irregularities in the lines were smoothed out, exhibiting the robustness of the Hough Transform method to missing data and to irregularities in the data. The higher level remaps show that longer lines indeed propagate to higher levels. However, there are also short lines that propagate up regardless of their lengths. These are the lines that lie across the boundaries of adjacent subimages that do not get grouped into one parent subimage until at a much higher level. All throughout the lower level



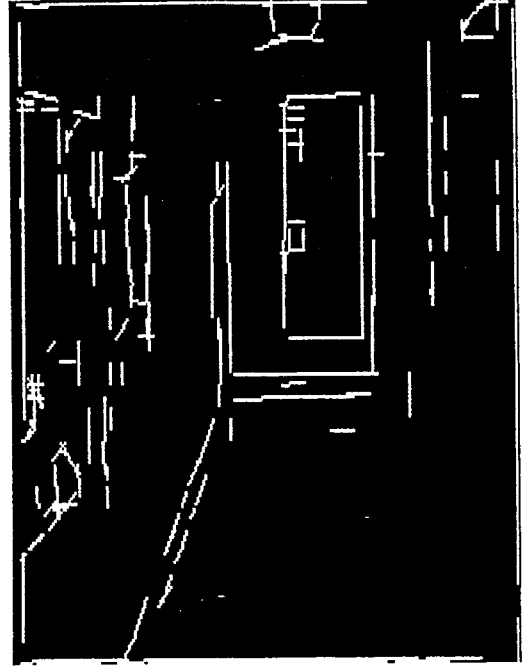
Figure 27. A binary line-thinned image.

groupings the part of the line that lies in one subimage consistently supports the other part resulting in the propagation of both parts to the level where their subimages become parts of a central subimage region. This may be attributed to the constraint that was set for the detectability of a line group.

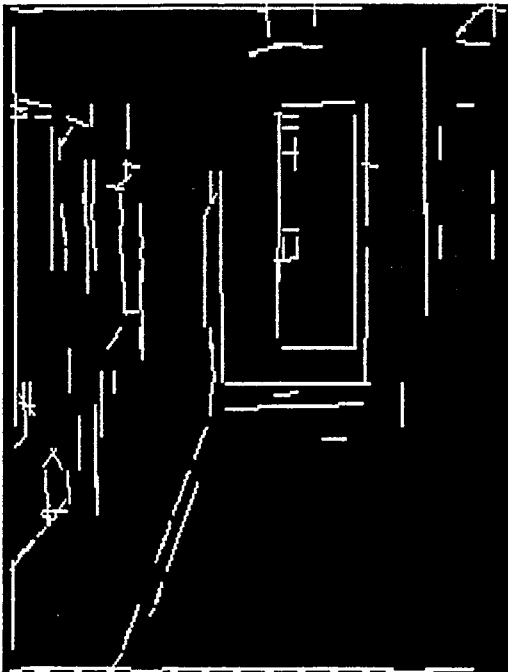
For this implementation, a line group is considered detectable or is indicated to exist if there are at least two feature points that are collinear, one feature point located in a central subimage and the other point lying in an immediate neighboring subimage which may be a central subimage or a sibling subimage outside of the central subimage region. Because of this constraint, the propensity of the Hough Transform method for combining feature points that are collinear in adjacent subimages into a single line regardless of their discontinuity in the lower level as exhibited in the formation of the level 5 lines.



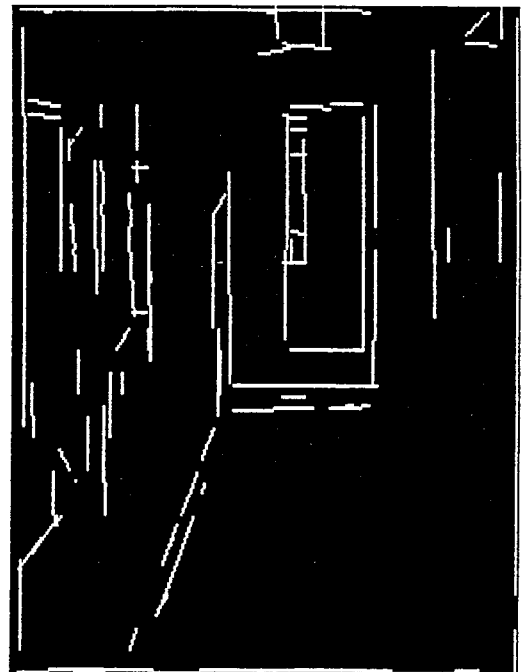
(a) Level 1



(b) Level 2

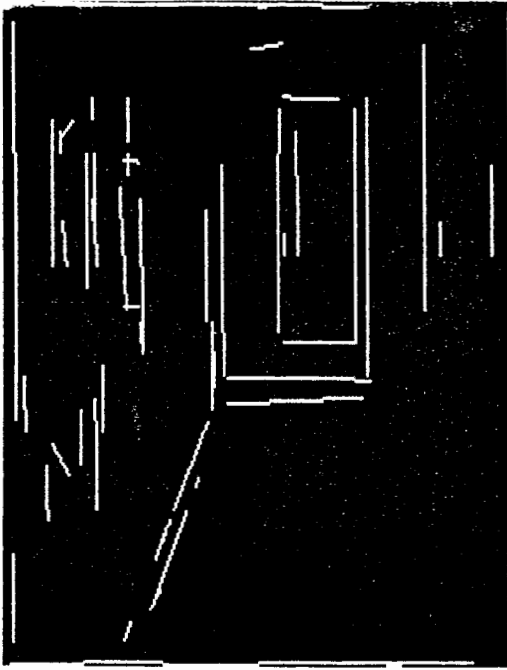


(c) Level 3

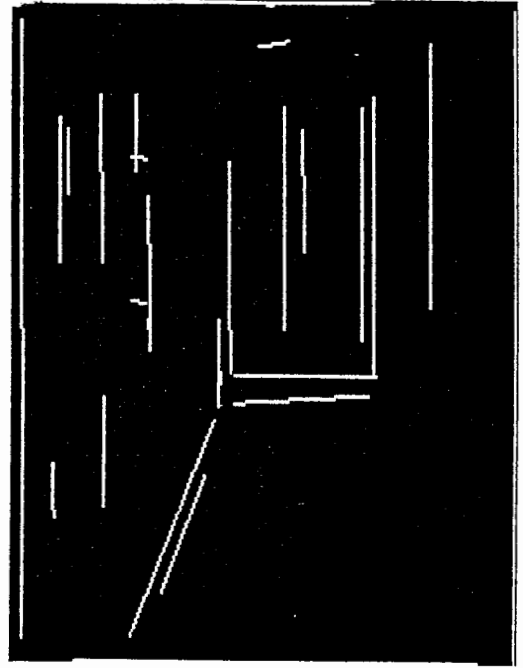


(d) Level 4

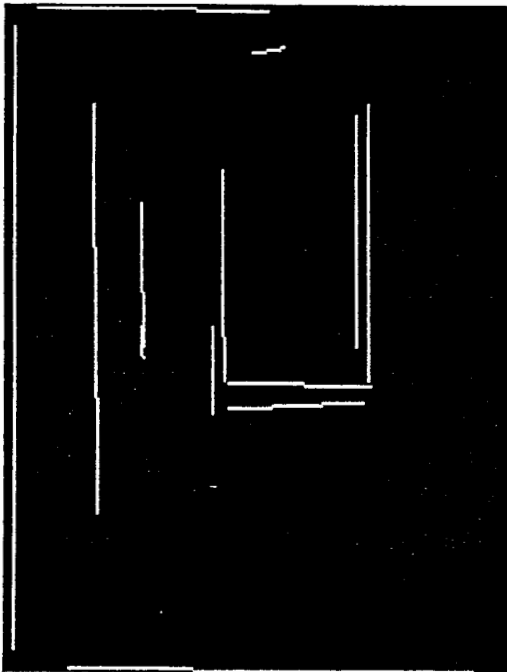
Figure 28. Remaps of the line segments detected at Levels 1, 2, 3 and 4 for image in Figure 27.



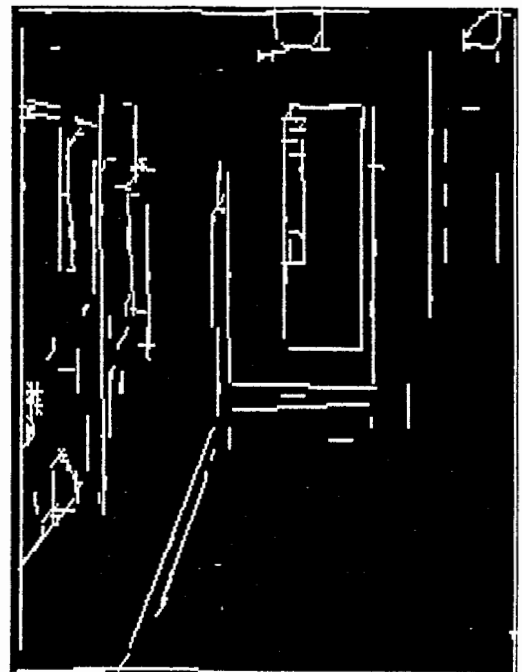
(a) Level 5



(b) Level 6



(c) Level 7



(d) All

Figure 29. Remaps of the line segments detected at Levels 5, 6, 7 and all the levels for image in Figure 27.

**IMPACT OF THE INITIAL SUBIMAGE SIZE
ON THE LINE FEATURE EXTRACTION**

The higher level grouping simply takes the line segments available in the lower level for consideration. It is obvious that the amount of detail that may be extracted highly depends on the lowest level line segment extraction from the input image. The amount of detail is determined by the initial subimage size into which the input image is partitioned, noting that the initial proximity constraint imposed by the circular region in the central subimage and the threshold for the shortest detectable line are determined by the subimage size. A smaller subimage size means a smaller accumulator array, which is a good point, but a smaller subimage size would also mean more grouping levels and more line segment elements to handle at each level. Table III presents the effects of the initial subimage size on the processing times (in seconds) for the lowest level line segment determination and the higher level groupings when the scheme was applied to the line-thinned images in Figure 30. The number of line segments identified at each level before and after the groupings are presented in Table IV. The results show that a smaller subimage size requires lesser processing time for the lowest level line segment determination, but incurs more time in the higher level groupings. The timing results indicate that the best subimage size is 8. The lowest level remaps for Image1 with initial subimage sizes 4, 8 and 16 are shown in Figure 31.

**TABLE III
PROCESSING TIMES FOR THE LINE FEATURE EXTRACTION AT
DIFFERENT INITIAL SUBIMAGE SIZES**

size	Image1			Image2		
	4	8	16	4	8	16
hhlow	31	51	101	35	64	130
hhhigh	53	15	5	66	24	11
total	84	66	106	101	88	141

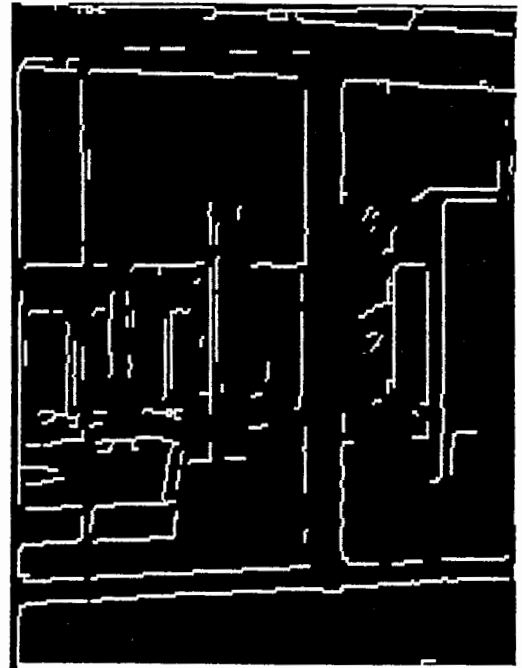
TABLE IV

NUMBER OF LINES EXTRACTED BY THE HIERARCHICAL SCHEME
AT DIFFERENT INITIAL SUBIMAGE SIZES

size	Image1						Image2					
	4		8		16		4		8		16	
level	i	f	i	f	i	f	i	f	i	f	i	f
1	1090	107	307	14	116	9	1379	57	470	21	211	9
2	562	34	174	12	63	2	729	30	254	7	112	10
3	328	37	99	14	35	3	413	24	141	13	63	3
4	177	31	51	9	19	3	222	25	80	9	37	7
5	88	17	27	5	8	8	114	15	47	14	15	15
6	44	13	11	11	-	-	60	14	16	16	-	-
7	14	14	-	-	-	-	20	20	-	-	-	-
total		253		65		25		185		80		44

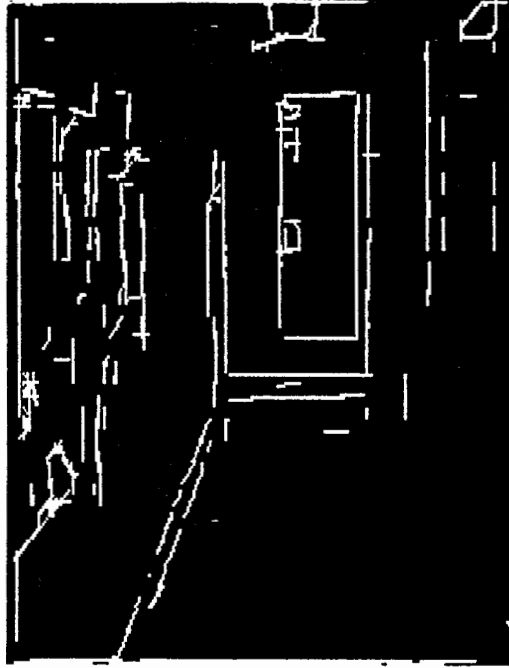


(a) Image1

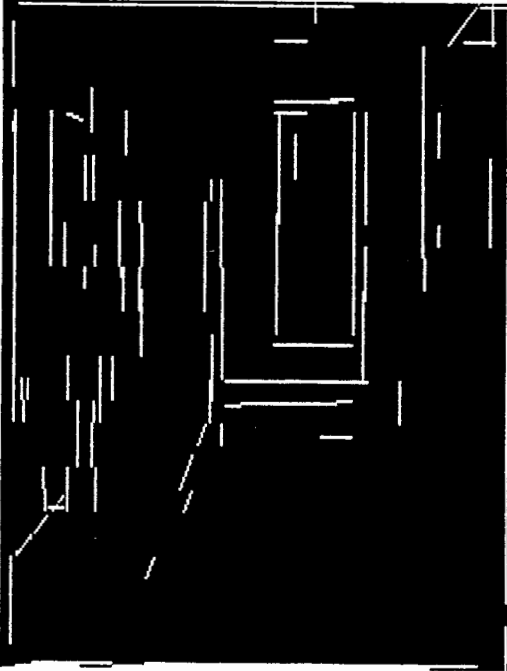


(b) Image2

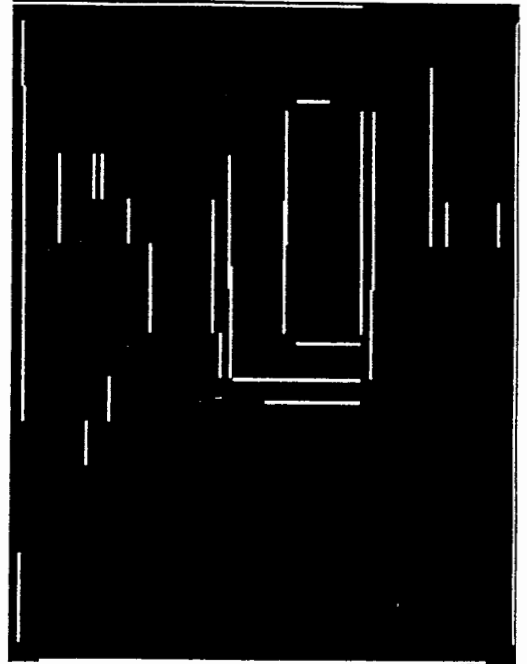
Figure 30. Binary line-thinned images used for initial subimage size analysis.



(a) size = 4



(b) size = 8



(c) size = 16

Figure 31. The Level 1 line segments detected with varied initial subimage sizes.

The remaps show that size 4 made the best extraction of the diagonal line defining the corridor outline, whereas a considerable part of this line was not detected with size 8 and the line was totally not visible with size 16. A clear outline of the corridor was propagated to level 6 (1 step from the top level) for the size 4 case, whereas a part of the corridor detected with size 8 was propagated to level 4 (2 steps from the top level) as shown in as shown in Figure 32.

Thus, size 4 was deemed to be a good initial subimage size despite the longer processing time involved considering the credibility of the features extracted and made visible at the higher levels.

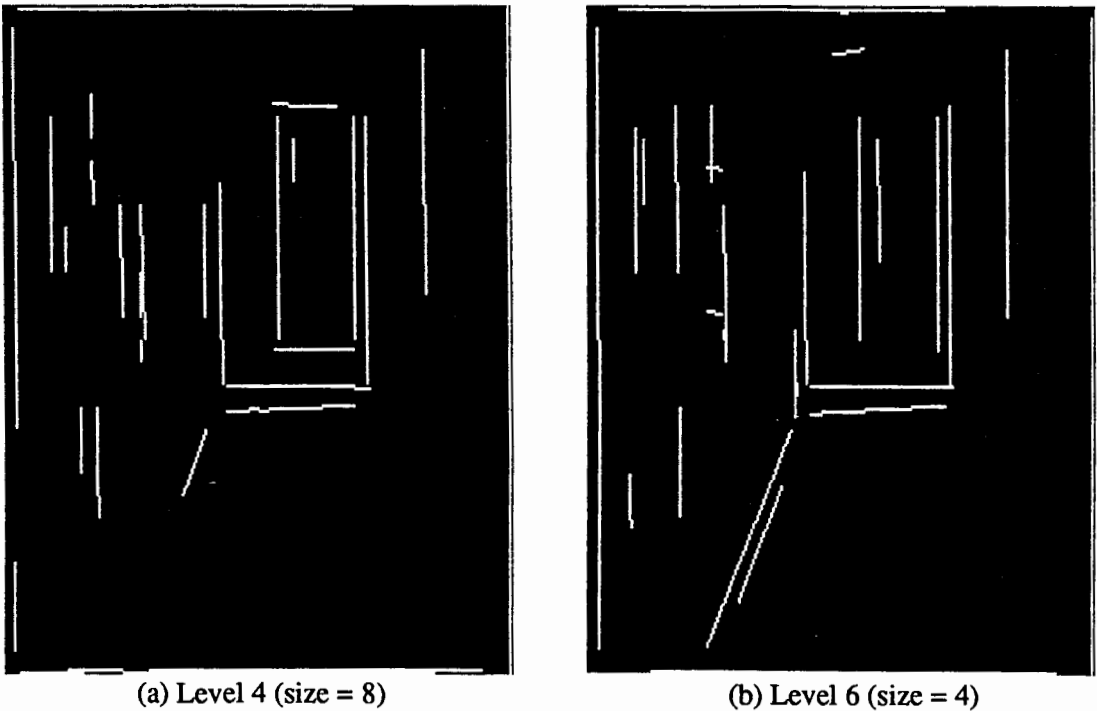


Figure 32. Detectability of the lines at higher levels.

FEATURE CHARACTERIZATION

The aforementioned grouping scheme maintains the hierarchical structure presented in Figure 24. At the end of the processing, each of the levels will only contain the lines (or groups of collinear segments) that were detected in the respective levels. The resulting structure may be viewed as lines grouped according to levels, where each level is a set of rows of lines and each row consisting of what may now be viewed as individual lines. However, each line is itself a group of short line segments obtained at the lowest level from the input image. At each level, a line is represented by its (ρ, θ) parameters referenced to the center of the subimage where the line was detected. The expanse of the subimage for each line is determined by the subimage size dictated by the hierarchical subimage grouping scheme imposed by the pyramidal treatment of the image.

A globalization of the lines' parameters affords a quicker view of the spatial relationship of all the lines regardless of their level classification. The global characterization involves: (1) expressing each line's parameters in terms of the complete image's center; (2) determination of the line's endpoints from the lowest level subsegments that make up the line; and (3) determination of the length of the line from these endpoints. With such a characterization, each line may be represented by its normal equation with the global (ρ, θ) as the parameters of the equation, and the line specifically located in the image by its endpoints.

Despite the globalization of the line parameters, the hierarchical grouping of the lines according to levels is preserved to allow the scene recognition subsystem to exploit the classification of the lines afforded by the hierarchical structure. To provide the scene recognition subsystem some general information about the scene, summaries of the contents of each level are made and these level summaries are combined to give an overall summary of the scene. A level summary consists of the length of the longest line, the number of horizontal lines, the number of vertical lines, the number of slanting lines (neither horizontal nor vertical) and the

total number of lines in the level. The scene summary therefore, consists of the length of the longest line in the scene, the total number of horizontal lines, the total number of vertical lines, the total number of slanting lines and the total number of lines found in the scene.

The program source codes for the postprocessing of the results generated by the hierarchical line extraction scheme, the summarization of the the said results and the generation of the final output is presented in Appendix I. This set of routines is called by the main Hierarchical Hough Transform Program given in Appendix E.

The characterizations and summaries are written in a file in a format that will enable the scene recognition subsystem to easily reconstruct the hierarchical structure for the scene recognition task. Each output line starts with a single letter to indicate the type of information each output line contains. An F indicates a comment line that reflects the meaning of each item in the data lines. A P indicates the overall summary that contains the number of levels in the hierarchy, the size of the image in terms of the bigger dimension, the length of the longest line found, the total number of horizontal lines, the total number of vertical lines, the total number of slanting lines and the overall total number of lines. L indicates the level summary, giving the level number, the length of the longest line in the level, the number of horizontal, vertical and slanting lines and the total number of lines found in the level. Entries for the description of the line segments found in each level start with an H, V or N for the horizontal line, vertical line or slanting line, respectively. A line segment entry contains the coordinates of the line's midpoint (y_s, x_s), the normal parameters of the line (ρ, θ), the length of the line and the expanse of the line ($\Delta y, \Delta x$). The normal parameters are given as integers, the values of which correspond to the interval in the quantized parameter space. The sizes of the quantization intervals $\rho_\Delta, \theta_\Delta$ are given in the entry denoted by C. The Δx and Δy items are only significant for slanting lines, but for the sake of uniformity in the representation, the values were assigned assigned as (256, 0) for vertical lines and (0, 256) for horizontal lines. A sample output listing is given in Appendix J.

CHAPTER VI

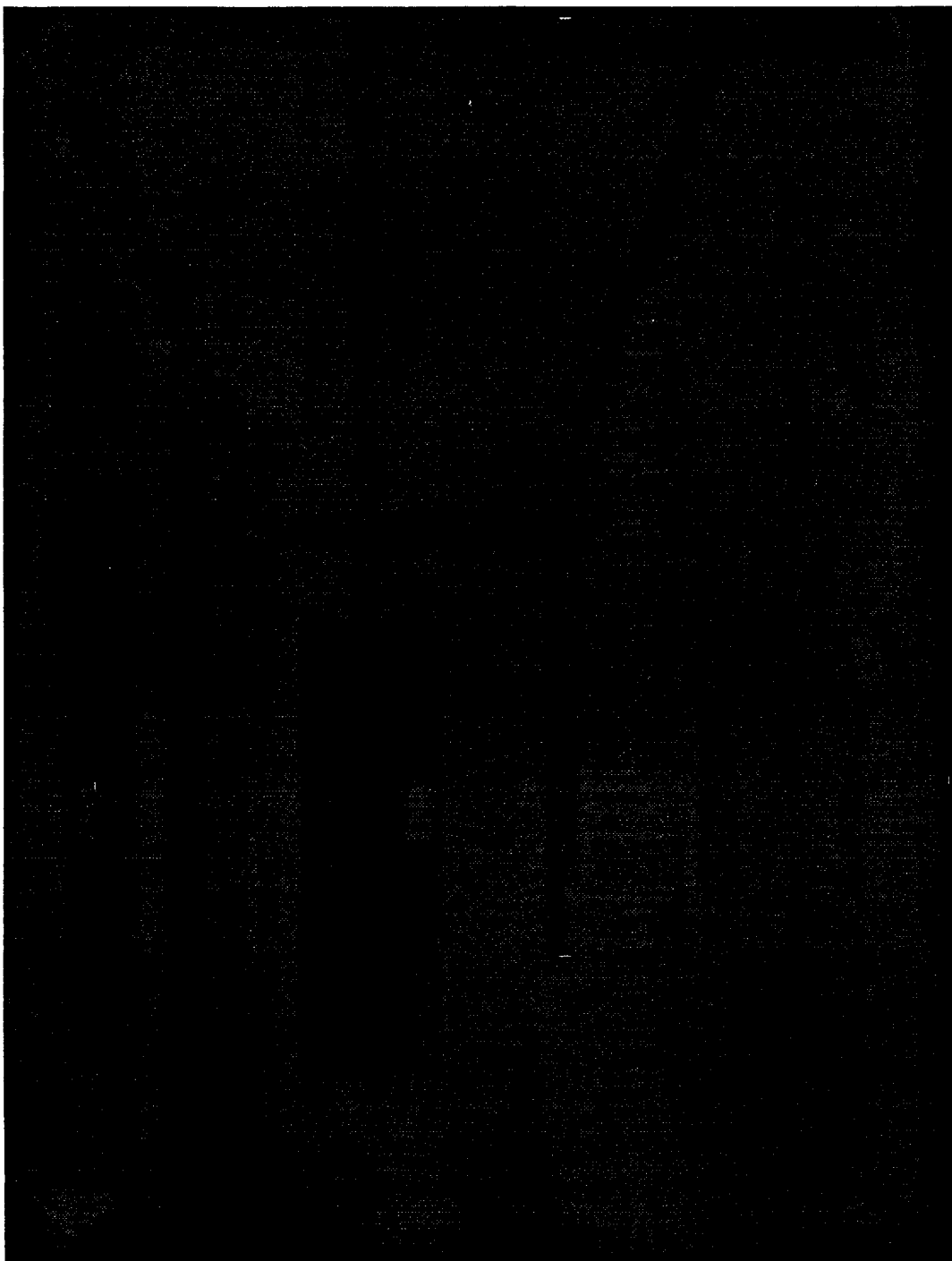
EVALUATION OF RESULTS

The sequence of processing originally conceived to be essential to the extraction of line features in indoor scenes consists of: Histogram Equalization, Smoothing with the use of the median filter, Edge Detection using the Sobel edge detectors, Binarization to extract the edges detected, Labelling, reBinarization and Thinning to refine the edges to thin lines and Line Extraction using a hierarchical approach to the Hough Transform method. The Binarization step includes the automatic determination of the binarization threshold from the gray level histogram of the image using the Between-Class-Variance method. The reBinarization step converts the labelled image back to the black and white state using a threshold of 1. Presented in Table V are the processing times in a PC-386SX for each of the operations when applied to Image1 shown in Figure 33.

TABLE V
EXECUTION TIMES FOR THE IMAGE PROCESSING
OPERATIONS ON IMAGE1

operation	time(sec)	% total time
Histogram Equalization	20	7.02
Smoothing	59	20.70
Edge Detection	50	17.54
Binarization	18	6.32
Labelling & reBinarization	22	7.72
Thinning	30	10.53
Line Extraction & Charac.	86	30.17
Total	285	100.00

Figure 33. Image1



A total processing time of 285 seconds is indeed unattractive for the real-time application envisioned for the system. To minimize the processing time, the necessity or dispensability of the operations is evaluated based on their impact on the scene description extracted and the processing times involved. Among the operations, Histogram Equalization, Edge Detection, Binarization, Thinning and Line Feature Extraction and Characterization are considered indispensable. However, Smoothing, Labelling and reBinarization are considered enhancement steps. Smoothing was used to reduce noise in the digitized image, hopefully to improve the detectability of the edges and to prevent the detection of false edges. Labelling and reBinarization are supposed to get rid of small spots or streaks in the binarized image, thereby reducing the number of pixels that will be processed by the Thinning and the Line Extraction steps. Histogram Equalization, Smoothing, Edge Detection and Binarization have constant execution times since the operations process all the image pixels regardless of the image quality or the density of edge features in the image. On the other hand, the processing times for Labelling, Thinning and Line Extraction are affected by the amount of pixels comprising the extracted edges. The evaluation is therefore centered on the examination of the factors that affect the execution time of the Thinning and the Line Extraction steps and the impact of the Smoothing and Labelling steps on the execution times and the resulting quality of the scene features extracted.

First and foremost, it was observed that the choice of a binarization threshold is a critical factor in the edge extraction steps. A relatively low threshold will yield thick edges, whereas a relatively high threshold might lose some edge details. Two automatic threshold determination methods were initially considered, namely, the Between-Class-Variance method (BCV) and the entropy-based method (ENT). As was earlier discussed in Chapter IV, two treatments of the gray level histogram of the gradient image were considered. One treatment includes the complete histogram for the threshold determination, whereas another treatment establishes the effective range of the histogram to include only the nonzero gradient pixels. The second treatment was deemed more appropriate since the main concern was the classification of the gradient

pixels as edge or non-edge pixels. As was pointed out, the threshold values based on the first treatment were lower than that of the second treatment. It was also noted that the ENT results are consistently much lower than the BCV results, an outcome that may be attributed to the criterion measures that were used as the basis for the separation of the classes. The ENT takes the point where the sum of the posteriori entropies of the two classes is maximum as the threshold point, which is tantamount to saying that it takes the point where the separation into the two classes will make probability distributions of the two classes have equal or at least similar. Whereas, the BCV considers the point where the variance between the classes is maximum as the threshold point. The binarized images in Chapter IV, show that the edges extracted by the threshold determined by ENT are much thicker than those of the BCV-determined values. Despite the higher values, the BCV-determined thresholds effectively extracted the edges. Anyway, the threshold values determined from both methods with the two treatments of the effective range of the histogram are hereby taken for determining the impact of the binarization threshold on the thinning process. The Table VI notations B0 and B1 denote BCV-determined thresholds taking the effective range of the histogram to start at gradient 0 and 1, respectively. Similarly, E0 and E1 denote entropy-based thresholds with the corresponding treatment of the histogram as B0 and B1, respectively. The entropy-based method gave the same threshold value E for the unsmoothed image considered.

The timing data for the thinning process presented in Table VI for Image2 in Figure 24 consistently show that the thinning times required for the binary images resulting from the threshold determined by the entropy-based method are consistently higher than that for the BCV-thresholded ones. The higher number of passes or iterations made in deleting the outer pixels of the edge streaks indicates more deletions made to finally skeletonize the edges.

Table VI also shows that although the thresholds for the unsmoothed image are practically the same as that for the smoothed image, the resulting binarized images take longer thinning times than their smoothed counterparts. The number of thinning passes are the same, indicating

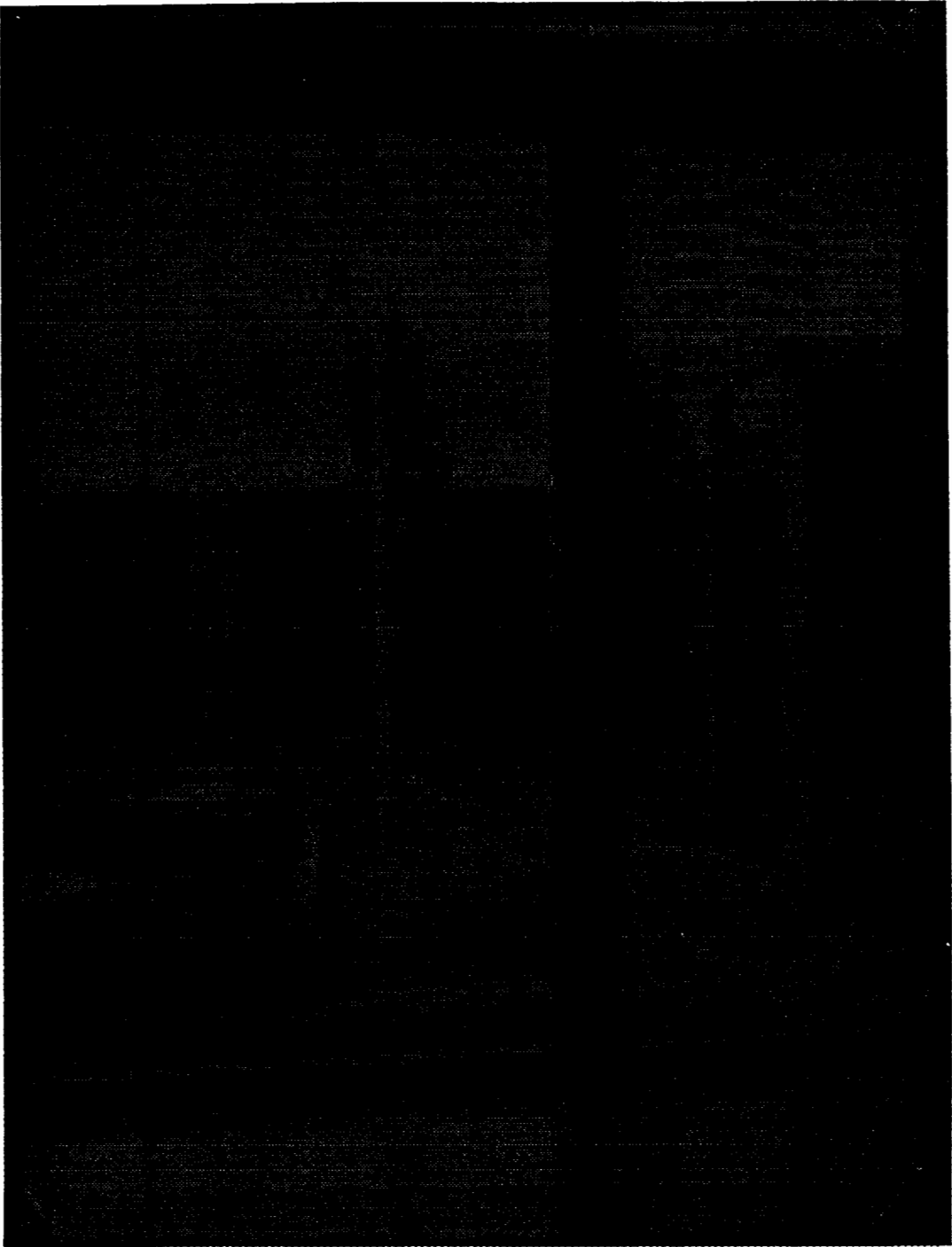


Figure 34. Image2

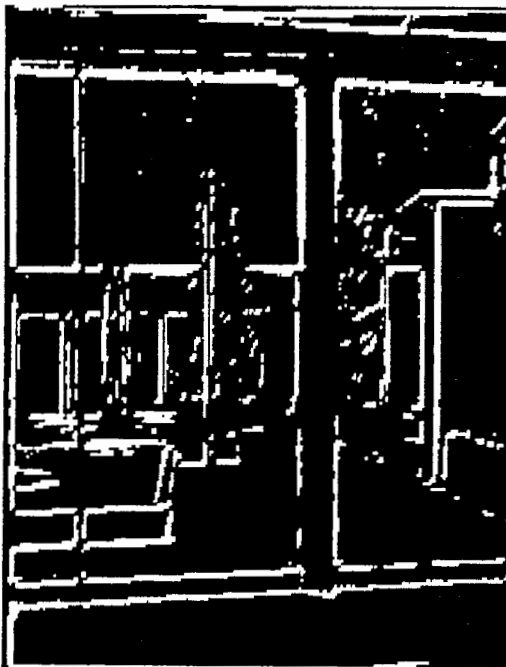
TABLE VI

EFFECT OF THE BINARIZATION THRESHOLD [T], SMOOTHING AND LABELLING ON THE THINNING TIME [s] AND NUMBER OF PASSES [P] FOR IMAGE2

	Smoothed Image								unSmoothed Image					
	with Labelling				no Labelling				with Labelling			no Labelling		
	B1	B0	E1	E0	B1	B0	E1	E0	B1	B0	E	B1	B0	E
T	93	86	75	68	93	86	75	68	93	86	78	93	86	78
s	31	32	37	37	31	32	37	38	32	33	38	33	33	40
P	3	3	4	4	3	3	4	4	3	3	4	3	3	4

that it is the higher number of edge pixels that caused the longer execution time rather than the thickness of the edge streaks. Figure 35 shows that there are more spots and streaks of white pixels in the binarized unsmoothed image than in the binarized smoothed one.

The effect of Labelling is not so discernable in the thinning data. However, a comparison



(a) smoothed image



(b) unsmoothed image

Figure 35. The effect of smoothing on binarized images.

of the thinned labelled image with the thinned unlabelled image showed that some ends of the thinned lines in the labelled image were diminished. The erosion of the ends is a side effect of the way the labelling routine treats pixel connectivity. The labelling routine takes side-adjacency as the only requirement for connectivity, thereby deleting even those small streaks that happen to be connected to bigger blobs by one vertex only.

The final impact of the smoothing and labelling processes are determined by their effects on the Line Feature Extraction step. Table VII shows their effects on the processing times and Table VIII shows the number of lines identified at each level before and after the line segment grouping for Image1.

The use of the Labelling step for the removal of small blobs or streaks that may thin out to very short lines which will not be detectable anyway, improves the total execution time for both the smoothed and unsmoothed images. The figures for the number of lines detected in the lowest level however shows that some lines were made undetectable by the Labelling step. The Labelling step's negligence of vertex-connectivity in adjacent blobs caused the erosion of the edges at the ends making the end segment too short to be detected or worse, completely losing the end segment. Also, weak edges that tend to appear as a series of aligned streaks are of course eliminated. So in a sense, the Labelling step causes some losses in the detectable lines especially if the extracted edges that will yield the line are rather weak.

TABLE VII

EFFECTS OF SMOOTHING AND LABELLING ON THE PROCESSING TIMES
OF THE LINE FEATURE EXTRACTION ROUTINES FOR IMAGE1

	Smoothed Image		unSmoothed Image	
	with Labelling	no Labelling	with Labelling	no Labelling
hhlow	31	33	36	41
hhhigh	53	59	84	95
hpost	2	2	2	2
total	86	94	122	138

TABLE VIII
EFFECTS OF SMOOTHING AND LABELLING ON THE
NUMBER OF LINES EXTRACTED FROM IMAGE1

Level	Smoothed Image				unSmoothed Image			
	with Labelling		no Labelling		with Labelling		no Labelling	
1	1090	107	1194	127	1459	149	1640	192
2	562	34	611	45	773	78	863	91
3	328	37	355	43	443	67	491	84
4	177	31	191	35	240	51	262	54
5	88	17	96	15	126	31	138	38
6	44	13	49	16	54	22	56	24
7	14	14	15	15	14	14	14	14
total	253		296		411		496	

Elimination of the Smoothing step increased the processing times the more because of the increased number of white pixels that need to be parameterized. The number of lines formed at the lowest level is notably much higher than those formed with the smoothed image. Since the Hough Transform is not concerned about connectivity of pixels as long the right number of pixels vote for a parameter pair, then, the noise spots which the cluster within a span of the initial subimage size are contributors to a line if the number is right. But as long as the lines formed by the noise spots are rather isolated from the lines indicated by the true edges, then there are no complications since these noise lines will never propagate into the higher levels. The problem arises when the noisy spots cluster around the edges and form lines with the real edge pixels. Remaps of the results show that the lines are indeed formed by the noise clusters and get linked with real lines during the grouping so that the "adulterated" line becomes longer and propagates to higher levels.

Because of the propensity of noise to gather around edges, and because of the high regard of the Hough Transform scheme for collinear points regardless of their connectivity, it is not really safe to disregard the initial noise removal from the processing sequence. However, the Labelling step is dispensable since the small pixel blobs which the labelling step removes are

discarded just the same by the lowest level line segment determination step. Although there is an increase in the line extraction step processing time, the increase is still less than the time incurred for the Labelling and the reBinarization steps. The tendency of the Labelling routine to cut off edges is indeed detrimental to the extraction of the desired features especially when the edges are rather weak because of poor image quality or inadequate lighting.

In the final analysis, Labelling may be eliminated as an edge enhancement step, thereby eliminating the need for reBinarization prior to Thinning. Smoothing, however, is to stay to ensure the credibility of the features extracted. Table IX presents a comparison of the execution times for the image processing steps if the sequence of operations includes Labelling and reBinarization (seq1) or excludes the Labelling and reBinarization (seq2). The elimination of the Labelling step had no effect on the Thinning time but it did increase the processing time for Line Feature Extraction and Characterization.

TABLE IX
EFFECTS OF LABELLING ON THE EXECUTION TIMES
FOR THE IMAGE PROCESSING OPERATIONS

	Image1		Image2	
	seq1	seq2	seq1	seq2
Image Enhancement	79	79	79	79
Edge Detection	50	50	50	50
Binarization	18	18	18	18
Labelling & reBinarization	22	-	23	-
Thinning	30	30	31	31
Line Extraction & Charac.	86	94	103	110
total	285	271	304	288

CHAPTER VII

CONCLUSIONS AND FUTURE WORK

CONCLUSIONS

The low to medium level image processing system developed in this thesis was basically designed to be the front end of the vision system of the PSUBOT. As such, the system must be simple, inexpensive and automatic to meet the transportability requirement of the autonomous wheelchair robot.

To devise the simplest possible system, the thesis started with the identification of the scene features that the scene recognition system might need for the recognition task and devised a sequencing of the minimal operations that will lead to the generation of the scene description in terms of these features. A thorough study of the image processing operations that favor the extraction of object boundaries in natural scene images was made and presented in this thesis. True to the spirit of engineering, published performance surveys and evaluations were used as the starting point for the selection, available software were used, and gaps were filled in to come up with a system that will effect a spontaneous processing of the image from the image acquisition step to the final generation of the scene description. To completely automatize the image preprocessing, programs for the unsupervised binarization threshold determination were written based on the entropic method and the between-class-variance method. Tests made on the real images used showed the between-class-variance method to be most appropriate for the application intended. The sequence of operations were finally evaluated for efficiency in effecting their respective functions on the natural scene images, effects on the performance of the other operations, processing speed and their ultimate impact on the credibility of the scene features

extracted. The final sequence of preprocessing steps consists of Histogram Equalization, Image Smoothing using the median filter, Edge Detection with the Sobel operator, Binarization using the automatically determined threshold, and Edge Thinning. The sequence is now fully automatic, effecting a spontaneous processing of the image from the image acquisition step to the generation of the line-thinned image that is input to the feature extraction steps.

The real crux of this thesis work, is the development of a hierarchical line feature extraction scheme that will provide the scene recognition system with a credible, easily reconstructable, and compact description of the scene. The hierarchical approach to line extraction used the basic concepts of the Hough Transform method and pyramids to offset the weaknesses of the Hough Transform method and to exploit its strong points towards a robust line feature extraction scheme. The "Hierarchical Hough Transform" program was written to implement the scheme. Parameters that affect the efficiency of the scheme in extracting the salient scene features were identified, but due to time constraints, only the initial subimage size was subjected to a thorough analysis. Tests on indoor scene images proved that using the appropriate initial subimage, the scheme can efficiently extract line feature descriptions even in images with relatively poor quality. The scheme proved robust to isolated noisy spots in the image and is tolerant of missing data, characteristics that may be attributed to the use of the Hough Transform method. However, unlike the conventional Hough Transform method implementation, accidental associations of edge points into false lines are reduced because of the localization of the transform to subimages. A natural classification of the features is effected by the hierarchical treatment, a classification that may be exploited by the scene recognition task.

In the final analysis, the thesis was successful in devising an automatic system that is completely implementable in a PC-386SX with 640K base memory, thereby satisfying the transportability requirements of the robot. Line feature extraction itself is a tedious process, requiring as much time and space as the image preprocessing operations. Although the idea of the Hough Transform method and pyramids are not really new and novel, the implementation of the line

feature extraction step in the PC and on real scenes at that, proves that there is always a way to go around the limitations of space and time.

The use of visual perception in autonomous mobile systems was approached with caution by most mobile system developers because of some inherent characteristics of image processing operations. Most image processing operations involve intensive calculations, intensive memory use, big memory requirements and are rather reputed to be slow. If ever used, the image processing operations are done on multiprocessors, more powerful computers or bigger systems, thereby requiring the mobile robot to be wired to the computer or be radio controlled from the main system or be loaded with the all these complexities. The fact that the system developed in this thesis is completely implementable in a PC that is perfectly transportable and inexpensive, is almost a breakthrough in the applicability of visual perception in mobile systems that may be used for the more mundane but noble applications like mobility aids for the handicapped and the elderly.

In the light of the evaluation criteria set forth for this endeavor, the system developed meets the transportability and efficacy requirements but is very slow for the real-time application intended. However, the thesis has at least devised a system that may serve as the core for further developments to improve the processing speed aspect. The thesis has also identified the parameters that may be manipulated to further enhance the overall performance of the system.

FUTURE WORK

The main concern in the development of the system was devising a scheme that will lead to the effective extraction of scene features for the PSUBOT's scene recognition system. Now that the whole processing system fits in the confines of the PC thereby satisfying transportability requirement, it is about time to improve the real-timeliness of the whole set-up. The nature of the processes chosen are highly amenable to parallel processing, but the use of multiprocessors unless implementable in a PC set-up will defeat one of the objectives of the undertaking.

Pipelining the operations may be considered so as to gain some headstart in the operations that needs to process a part of the image at a time, independent of the other parts. This might be used for the Smoothing, Edge Detection, and the lowest level line segment determination.

The line extraction scheme developed in this thesis has a number of offshoots. Variations of the proximity and collinearity constraints may be tried and analyzed to improve its tolerance to noise introduced by the acquisition process. Though it was proven to perform well in images where the noise spots are rather isolated from the edges, it would be an added gain if the smoothing process be eliminated altogether. The constraints that were used in this thesis were rather lax thereby propagating short lines to higher levels. Some selectivity with respect to line length may be tried so as to give a better classification of the lines into levels according to their relative lengths. This in effect will improve noise tolerance since the noise lines that are not necessarily attached to real lines will not be propagated to higher levels thereby preventing their association with the real lines. The lines left in the lowest level may then be treated as the noisy lines and be simply discarded.

The concept of the hierarchical approach and the Hough Transform may be extended to the extraction of nonlinear features. If the conventional Hough Transform method is applicable to nonlinear curves, then there must be a way of detecting these nonlinear curves in a hierarchical manner. As it did with the linear features problem, the hierarchical approach will surely reduce the amount of parameter space that need to be examined at a time thereby reducing the accumulator size problem that indeed increases with the increased dimensionality of the parameterizations.

REFERENCES

- [1] Gonzalez, Rafael C. and Paul Wintz, *Digital Image Processing*, Addison-Wesley Publishing Company, Massachusetts, 1979.
- [2] Ballard, Dana H. and Christopher M. Brown, *Computer Vision*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632, 1982.
- [3] Nagel, Roger N., "State of the Art and Predictions for Artificial Intelligence and Robotics", *Robotics and Artificial Intelligence*, NATO ASI Series Vol. F11, Springer-Verlag Berlin Heidelberg, 1984, pp. 3-45.
- [4] Giralt, Georges, "Mobile Robots", *Robotics and Artificial Intelligence*, NATO ASI Series Vol. F11, Springer-Verlag Berlin Heidelberg, 1984, pp. 365-393.
- [5] Albus, James S., "Robotics", *Robotics and Artificial Intelligence*, NATO ASI Series Vol. F11, Springer-Verlag Berlin Heidelberg, 1984, pp. 65-93.
- [6] Brady, Michael, "Artificial Intelligence and Robotics", *Robotics and Artificial Intelligence*, NATO ASI Series Vol. F11, Springer-Verlag Berlin Heidelberg, 1984, pp. 47-63.
- [7] Albus, James S., "Robotics: where has it been? Where is it going?", *Robotics and Autonomous Systems*, Vol. 6, No. 3, July 1990, pp. 199-219.
- [8] Ehtashami, Mohammad, Sung J. Oh and Ernest L. Hall, "Omnidirectional Position Location for Mobile Robots", *Intelligent Robots and Computer Vision, Proceedings of SPIE - The International Society for Optical Engineering*, (eds: David P. Casanent and Ernest L. Hall), Vol. 521, 1984, pp. 62-73.
- [9] Stanton, Kevin B., Paul R. Sherman, Mark L. Rohwedder, Christopher P. Fleskes, David L. Gray, Dinh T. Minh, Cecilia Espinosa, Dieudonne Mayi, Mohammed Ishaque, and Marek A. Perkowski, "PSUBOT - A Voice-Controlled Wheelchair for Handicapped", *Proceedings of the International Midwest Symposium on Circuits and Systems*, August 1990, pp. 669-672.
- [10] Madarasz, Richard L. and Loren C. Heiny, "Visual Navigation Techniques for Indoor Transport Systems", A paper submitted by the Computer Science Department, Arizona State University, Tempe, Arizona 85287, to the 2nd World Congress on Robotics Research, 1985.
- [11] Dawson, Benjamin M., "Introduction to Image Processing Algorithms", *BYTE*, March 1987, pp. 169-186.
- [12] Dawson, Benjamin M., "Changing Perceptions of Reality", *BYTE*, December 1989, pp. 293-304.

- [13] Madarasz, Richard L., Loren C. Heiny, Robert F. Crompt and Neal M. Mazur, "The Design of an Autonomous Vehicle for the Disabled", *IEEE Journal of Robotics and Automation*, Vol. RA-2, No. 3, September 1986, pp. 117-126.
- [14] Napper Stan A. and Ronald L. Seaman, "Applications of Robots in Rehabilitation", *Robotics and Autonomous Systems*, Vol. 5, No. 3, November 1989, pp. 227-239.
- [15] Kunt, Murat, "Acquisition and Visualization", *Fundamentals in Computer Vision*, (ed. O.D. Faugeras), Cambridge University Press, Cambridge, 1983, pp. 1-26.
- [16] The PCVISIONplus Frame Grabber User's Manual, Part Number 47-H00010-01, Imaging Technology Inc., Massachusetts, April 1987.
- [17] Legate, Alvin M., EE 406 Project Report submitted to Dr. Marek A. Perkowski, Department of Electrical Engineering, Portland State University, Spring 1989.
- [18] Gonzalez, Rafael C., "Image Enhancement and Restoration", *Handbook of Pattern Recognition and Image Processing*, (eds: Young, Tzay Y. and King-Sun Fu), Academic Press, Inc., Orlando, Florida, 1986, pp. 191-213.
- [19] Levialdi, S., "Edge Extraction Techniques", *Fundamentals in Computer Vision*, (ed. O.D. Faugeras), Cambridge University Press, Cambridge, 1983, pp. 117-144.
- [20] Nevatia, R., "Image Segmentation", *HandBook of Pattern Recognition and Image Processing*, (eds: Young, T. and K.S. Fu), Academic Press, Inc., Orlando, Florida, USA, 1986, pp. 215-231.
- [21] Zhang, T.Y. and C.Y. Suen, "A Fast Parallel Algorithm for Thinning Digital Patterns", *Communications of the ACM*, Vol. 27, No. 3, March 1984, pp. 236-239.
- [22] Otsu, Nobuyuki, "A Threshold Selection Method from Gray-Level Histograms", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-9, No. 1, January 1979, pp. 62-66.
- [23] Cooper, George R. and Clare D. McGillem, *Probabilistic Methods of Signal and System Analysis*, CBS College Publishing, New York, 1986.
- [24] Bovik, Alan Conrad, Thomas S. Huang and David C. Munson, Jr., "The Effect of Median Filtering on Edge Estimation and Detection", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-9, No. 2, March 1987, pp. 181-194.
- [25] Kitchen, L.J. and J.A. Malin, "The Effect of Spatial Discretization on the Magnitude and Direction Response of Simple Differential Edge Operators on a Step Edge", *Computer Vision, Graphics, and Image Processing*, Vol. 47, No. 2, August 1989, pp. 243-258.
- [26] Kapur, J.N., P.K. Sahoo and A.K.C. Wong, "A new Method for Gray-Level Picture Thresholding Using the Entropy of the Histogram", *Computer Vision, Graphics, and Image Processing*, Vol. 29, No. 3, March 1985, pp. 273-285.

- [27] Weszka, Joan S. and Azriel Rosenfeld, "Histogram Modification for Threshold Selection", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-9, No. 1, January 1979, pp. 38-52.
- [28] Pun, T., "Entropic Thresholding, A New Approach", *Computer Vision, Graphics, and Image Processing*, Vol. 16, No. 3, July 1981, pp. 210-239.
- [29] Abutaleb, Ahmed S., "Automatic Thresholding of Gray-Level Pictures Using Two-Dimensional Entropy", *Computer Vision, Graphics, and Image Processing*, Vol. 47, No. 1, July 1989, pp. 22-32.
- [30] Sahoo, P.K., S. Soltani, and A.K.C. Wong, "A Survey of Thresholding Techniques", *Computer Vision, Graphics, and Image Processing*, Vol. 41, No. 2, February 1988, pp. 233-260.
- [31] Lee, Sang Uk and Seok Yoon Chung, "A Comparative Performance Study of Several Global Thresholding Techniques for Segmentation", *Computer Vision, Graphics, and Image Processing*, Vol. 52, No. 2, November 1990, pp. 171-190.
- [32] Kahn, P., L. Kitchen and E.M. Riseman, "A Fast Line Finder for Vision-Guided Robot Navigation" *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 12, No. 11, November 1990, pp. 1098-1102.
- [33] Burns, J. Brian, Allen R. Hanson and Edward M. Riseman, "Extracting Straight Lines", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-8, No. 4, July 1986, pp. 425-455.
- [34] Nevatia Ramakant and K. Ramesh Babu, "Line Feature Extraction and Description", *Computer Graphics and Image Processing*, Vol. 13, No. 3, July 1980, pp. 257-269.
- [35] Hung, S.H.Y. and T. Kasvand, "Linear Approximation of Quantized Thin Lines", *Pictorial Data Analysis*, (ed: Robert M. Haralick), NATO ASI Series Vol. F4, Springer-Verlag Berlin Heidelberg, 1983, pp. 15-28.
- [36] Shneier, Michael O., "Extracting Linear Features from Images Using Pyramids", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-12, No. 4, July/August 1982, pp. 569-572.
- [37] Princen, John, John Illingworth and Josef Kittler, "A Hierarchical Approach to Line Extraction Based on the Hough Transform", *Computer Vision, Graphics, and Image Processing*, Vol. 52, No. 1, October 1990, pp. 57-77.
- [38] Shneier, Michael, "Two Hierarchical Linear Feature Representations: Edge Pyramids and Edge Quadrees", *Computer Graphics and Image Processing*, Vol 17, No. 3, November 1981, pp. 211-224.
- [39] Rosenfeld, A., "Quadrees and Pyramids: A hierarchical representation of images", *Pictorial Data Analysis*, (ed: Robert M. Haralick), NATO ASI Series Vol. F4, Springer-Verlag Berlin Heidelberg, 1983, pp. 29-42.

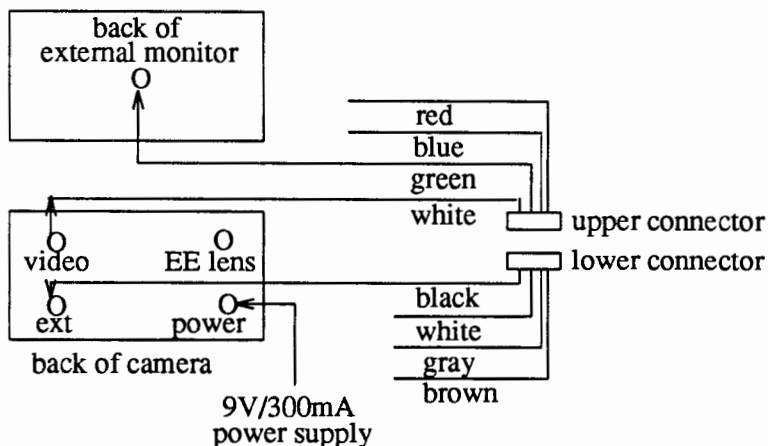
- [40] Hong, Tsai-Hong, Michael Shneier and Azriel Rosenfeld, "Border Extraction Using Linked Edge Pyramids", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-12, No. 5, September/October 1982, pp. 660-668.
- [41] Duda, Richard O. and Peter E. Hart, "Use of the Hough Transformation to Detect Lines and Curves in Pictures", *Communications of the ACM*, Vol. 15, No. 1, January 1972, pp. 11-15.
- [42] Illingworth, J. and J. Kittler, "A Survey of the Hough Transform", *Computer Vision, Graphics, and Image Processing*, Vol. 44, No. 1, October 1988, pp. 87-116.
- [43] Van Veen, T. M. and F.C.A. Groen, "Discretization Errors in the Hough Transform", *Pattern Recognition*, Vol. 14, 1981, pp. 137-145.
- [44] Gerig, Guido and Fernan Klein, "Fast Contour Identification through Efficient Hough Transform and Simplified Interpretation Strategy", *Proceedings: Eighth International Conference on Pattern Recognition*, Paris, France, 1986, pp. 498-500.

APPENDIX A

SET-UP FOR IMAGE ACQUISITION

The image acquisition hardware components consists of the PCVISIONplus FRAME GRABBER card that plugs directly into an expansion slot in the PC, a CCD camera that provides the video input, and an optional external video monitor to display the acquired images. The card contains the digitizer for the incoming video signal from the camera, the image memory where the resulting pixels (digital values) are stored, and display logic that converts the pixels in the frame memory back to an analog RS-170 format for display on the external monitor.

The card is provided with two rectangular 8-pin connectors that protrude thru the back panel of the PC into which two cables are plugged in to connect the card to the camera and the external monitor. Each cable has a square female connector at one end that plugs into the 8-pin connector and 4 BNC connectors on the other end, each of the BNC connectors color coded according to its function. The standard cable plugs into the upper connector, and the optional sync cable plugs into the lower connector. The connections for the image acquisition hardware components are as shown:



Details of the jumper locations and settings for the image memory, registers, and input video synchronization modes are given in the PCVISIONplus FRAME GRABBER User's Manual (Part No. 47-H00010-01).

During image acquisition the frame grabber drives the camera with composite sync via the black BNC connector of the optional sync cable that must be connected to the EXT socket of the camera. The frame grabber receives the composite video input via the white BNC connector of the standard cable which is connected to the VIDEO socket of the camera. The camera is provided with a 9v/300mA power adaptor. The green BNC connector of the standard cable is connected to the external monitor to give a monochrome video output signal with optional sync. The other BNC connectors are used for true color systems and other synchronization modes and are not used in this application.

The image acquisition is initiated by the acquire routine in SIMPP2 which provides 2 modes of acquiring the image. The snap mode acquires a single image from the video source. The grab mode acquires a continuous live image from the video source. Entering an integer during the grab mode terminates the acquisition process, thereby recording the last frame grabbed in the image memory. The image is displayed on the external monitor during the acquisition process.

APPENDIX B

THE PROGRAM SOURCE CODE FOR THE BINARIZATION
THRESHOLD DETERMINATION USING THE
BETWEEN-CLASS-VARIANCE METHOD

```

/*****
* s2thrbcv.c
* binarization threshold determination using the between-class-variance
* method of Nobuyuki Otsu from "A Threshold Selection Method from Gray-
* Level Histogram", IEEE Trans. on Systems, Man, and Cybernetics,
* Vol. SMC-9, No. 1, January 1979, pp. 62-66.
*
* INPUT : start -- desired starting point for effective range of
*         histogram
*         freq[] -- gray level histogram determined by the SIMPP2
*         histogram determination routine
* OUTPUT : threshval - the binarization threshold value.
*****/
#include "simpp2.h"

int bc_var(int start, long freq[])
{
    int long ftotal;      /* total frequency */
    float w[PIXEL_SIZE]; /* cumulative probability of class occurrence */
    float u[PIXEL_SIZE]; /* cumulative class mean level */
    double bcv, bcvmax=0.0; /* between-class variance, the maximum bcv */
    double tvar=0.0;     /* total variance */
    double ut;          /* total mean level */
    float ptemp;        /* probability, Pr(i) = freq[]/N */
    int threshval;     /* the threshold value */
    int end;           /* end of effective range of histogram */
    int i;             /* general loop variable */

    /* get total freq and establish effective range of histogram */
    /* start at the first nonzero freq from the start value specified */
    while ((start < PIXEL_SIZE) && (freq[start] == 0L))
        start++;
    if (start == PIXEL_SIZE)
    { printf ("histogram is empty\n");
      return (0);
    }
    /* find the end of the effective range of the histogram */
    end = start;
    ftotal = freq[start];
    for (i=(start+1); i<PIXEL_SIZE; i++)
        if (freq[i] > 0L)
        { end = i;
          ftotal += freq[i];
        }
    if (start == end)
    { printf ("there's only one gray level: %ld\n", start);
      return (start+1);
    }
}

```

```

/* determine cumulative probability, w[u]; cumulative class mean, u[] */
ptemp = ((float)freq[start]) / (float)ftotal;
w[start] = ptemp;          /* probability of class occurrence, Pr(i) */
u[start] = start * ptemp;  /* class mean level, iPr(i) */
for (i=(start+1); i<(end+1); i++)
{ ptemp = ((float)freq[i]) / (float)ftotal;
  w[i] = w[i-1] + ptemp;    /* cumulative Pr(i) */
  u[i] = u[i-1] + i*ptemp;  /* cumulative iPr(i) */
}
ut = (double)u[end];      /* total mean level */

/* determine total variance */
for (i=start; i<(end+1); i++)
  tvar = tvar + (i-ut)*(i-ut)*(((double)freq[i])/((double)ftotal));

/* compute bcv, and search for the maximum value */
i = start;
while (i<end)
{ bcv = (ut * w[i]) - u[i];
  bcv = (bcv*bcv) / ((double)w[i] * ((double)1 - w[i]));
  if (bcv >= bcvmax)
  { bcvmax = bcv;
    threshval = i;
    i++;
  }
  else
    break; /* the last bcv is the maximum */
}
return (threshval+1);
} /* end threshold */

/***** end s2thrbcv.c *****/

```

APPENDIX C

THE PROGRAM SOURCE CODE FOR THE BINARIZATION THRESHOLD
DETERMINATION USING THE ENTROPY-BASED METHOD

```

/*****
* s2thrent.c
* threshold determination using one-dimensional entropy of histogram.
* based on the entropy-based method as improved by Kapur, Sahoo and Wong,
* "A new method for gray-level picture thresholding using the entropy of
* of the histogram", Computer Vision, Graphics, and Image Processing,
* Vol. 29, No.3, March 1985, pp.273-285.
*
* INPUT : start -- desired starting point for the effective range of the
*         histogram
*         freq[] - gray level histogram determined by the SIMPP2 histogram
*         determination routine
* OUTPUT: threshval - threshold value
*****/
#include "simpp2.h"
#include <math.h>
#include <alloc.h>

int entropy(int start, long freq[])
{
    long ftotal;      /* sum total of all pixel values */
    double pf[PIXEL_SIZE]; /* probability of occurrence of gray level */
    double *Hs;      /* cumulative entropy at the particular pixel val */
    double pfm, Hm;  /* cumulative probability, entropy at end of freq range*/
    double psi, maxpsi; /* the criterion measure */
    int threshval;   /* the threshold value (at maxpsi) */
    int end;         /* end of the effective histogram range */
    int i;           /* general loop variable */

    /* get total freq and establish effective range of histogram */
    /* find the first nonzero freq --> start of effective range */
    while ((start < PIXEL_SIZE) && (freq[start] == 0L))
        start++;
    if (start == PIXEL_SIZE)
    { printf("histogram is empty.\n");
      return (0);
    }
    /* find the end of the effective range of the histogram */
    end = start;
    ftotal = freq[start];
    for (i=(start+1); i<PIXEL_SIZE; i++)
        if (freq[i] > 0L)
        { end = i;
          ftotal += freq[i];
        }
    if (start == end)
    { printf("there's only one gray leve\n");
      return (start+1);
    }
}

```

```

/* allocate space for the cumulative entropy values */
Hs = (double *)malloc(PIXEL_SIZE*sizeof(double));
if (Hs == NULL)
{ printf("\nmalloc failed for Hs allocation.\n");
  exit (1);
}

/* pf[] : probability of occurrence of specific gray level */
/* cumulative entropy Hs = sum of (pf log(pf)) from i=start to i=s */
pf[start] = Hs[start] = (double)0;
pf[start] = ((double)freq[start])/((double)ftotal);
Hs[start] = -(pf[start]) * log(pf[start]);
pfm = pf[start];
Hm = Hs[start];
for (i=(start+1); i<(end+1); i++)
{ pf[i] = Hs[i] = (double)0;
  pf[i] = ((double)freq[i])/((double)ftotal);
  if (pf[i] == 0.0)
    Hs[i] = Hm;
  else
    Hs[i] = Hm - pf[i] * log(pf[i]);
  pf[i] = pfm + pf[i]; /* convert probability to cumulative probability */
  pfm = pf[i];
  Hm = Hs[i];
}

/* search for maximum entropy */
i = start;
maxpsi = 0;
while (i<end)
{ psi = log(pf[i]*(1- pf[i])) + Hs[i]/pf[i] + (Hm-Hs[i])/(1-pf[i]);
  if (psi >= maxpsi)
  { maxpsi = psi;
    threshval = i;
    i++;
  }
  else
    break; /* a maximum entropy was found */
}
free(Hs);
return (threshval+1);
} /* end entropy */
/***** end s2thrent.c *****/

```

APPENDIX D

DATA STRUCTURE DEFINITIONS FOR THE HIERARCHICAL REPRESENTATION OF LINE FEATURES

```

/*****
* s2hhdefs.h
* global definitions for hough routines (hierarchical HT approach)
*****/
#define PI 3.1416

/* input image specs for buffers and array declarations */
#define DXC 256 /* x dimension (width of image in pixels) */
#define DYC 240 /* y dimension (length of image in pixels) */
#define SUBSIZE 16 /* maximum initial subimage size that may be used */
#define GSIZE 2 /* size of central subimage region == 2x2 neighborhood */

/***** structures *****/
struct sincos { /* sincos look-up-table struct */
    double sint;
    double cost;
};
typedef struct sincos scpair;

struct ptrs { /* subimage bounds pointers */
    int yklead; /* extra rows at top and bottom for mod2 square fit */
    int xklead; /* extra cols at left and right for mod2 square fit */
    int jstart; /* buffer row that contains an image row */
    int jend; /* buffer row last filled */
    int istart; /* first window that includes image */
    int iend; /* end of window that contains image */
    int ykstart; /* first subimage row in buffer in image coordinates */
    int ykend; /* last subimage row in buffer in image coordinates */
    int ykcentral; /* start of central subimage row in buffer */
    int xkstart; /* first image column in window in image coordinates */
    int xkend; /* last image column in window in image coordinates */
    int xkcentral; /* start of central subimage column in buffer */
    float ys; /* local center of window in image coordinates */
    float xs;
};
typedef struct ptrs bounds;

struct pt { /* feature point coordinates */
    float xp;
    float yp;
    struct pt *next;
};
typedef struct pt featpt;

struct line { /* line segment local parameters */
    int len; /* group membership indicator/ final line length */
    float xs; /* x of local center of subimage containing line segment */
    float ys; /* y of local center of subimage containing line segment */
    int rhos; /* normal distance of segment from local center (accu.space)*/
    int thetas; /* angle of normal with horizontal x-axis (in accu. space) */
    struct line *next; /* pointer to the next line segment */
};

```



```

    featpt *subseg; /* for subsegment list */
};
typedef struct line lineseg;

struct rmark { /* rowmarker node */
    float ys; /* y's of subimage centers in row */
    lineseg *row; /* points to the row's line segments */
    struct rmark *nextrow; /* points to next row of subimages */
};
typedef struct rmark rowmark;

struct lmark { /* levelmarker node */
    int levelnum; /* level number */
    int levelsub0; /* size of central subimage region (in pixels) */
    double levelrhodel; /* rho-quant. interval for the level's HT space */
    double levelthetadel; /* theta-quant. interval for the level's HT space */
    rowmark *level; /* pointer to the level components (see Figure 24 of text)*/
    struct lmark *nextlevel; /* pointer to the next lower level */
};
typedef struct lmark levelmark;

struct anode { /* one cell in the HT accu. used for higher level grouping */
    int rhos; /* rho value in terms of the HT accumulator space */
    float vote; /* number of votes accumulated for the cell */
    int voter[GSIZE*4 +1]; /* to contain labels of voting central feat pt */
};
typedef struct anode acell;

/*****end s2hdefs.h*****/

```

APPENDIX E

THE MAIN PROGRAM SOURCE CODE FOR THE HIERARCHICAL
LINE EXTRACTION SCHEME USING THE HOUGH
TRANSFORM METHOD (HHOUGH)

```

/*****
* hough.c
* The Hierarchical Hough Transform Program
* a hierarchical approach to the Hough Transform method for the detection
* and extraction of straight lines by normal parameterization.
*****/
#include "simpp2.h"
#include "s2hdefs.h"
#include <math.h>
#include <alloc.h>

/* functions called */
void SetHTspace(void);
void CenterImage(int dx, int dy);
void SetSIbounds(void);

/* in hhlw.c */
rowmark *hhlw(int x, int y, int dx, int dy);

/* in hhigh.c */
rowmark *hhigh(levelmark *pyramid, acell *HTgrp[]);

/* in hhpost.c */
void hhpost(int dx, int dy, levelmark *pyramid);
void SummarizeLevs(levelmark *pyr);
levelmark *FreePyramid(levelmark *pyr);

/* global variables */
int sub0;          /* subimage size */
bounds sb;        /* subimage bounds markers */
int rhosize, thetasize; /* rho-theta space size */
int rhozero;      /* rho=0 position in rho-translated HTspace */
double rhodel, thetadel; /* rho-theta sampling intervals */
double delrho, deltheta; /* rho-theta error limits */
int threshnum;    /* min feature points reqd for a true line */
int curlevel;     /* current pyramid level */
int rhocells;     /* width of voting list (higher level grouping) */
double levrdel, levtdel; /* lev's rhodel, thetadel for accuvals->parvals */

/*****
*****
| main routine
| call lowlevel linesegment extraction from the subimages comprising the
| binary line-thinned image;
| create each higher level of the pyramid from the collinear linesegments
| of the lower level;
| call postprocessing routine to adjust line parameters to be referenced
| on the image center and to determine the final line attributes:
| midpoint, length, rho-theta parameters.
| INPUT : image location in image memory
| starting at (x,y) with dx columns, dy rows

```

```

*****
*****/
void hthough(int x, int y, int dx, int dy)
{ int i, j;
  rowmark *levhead;          /* row marker node pointer */
  levelmark *pyrlev, *pyramid; /* level node pointers */
  acell *HTgrp[GFSIZE*2 + 1]; /* group voting accumulator */

  /*** printf("initial pyramid level marker set\n"); ***/
  pyramid = (levelmark *)malloc(sizeof(levelmark));
  if (pyramid == NULL)
  { printf("malloc failed in pyramid level marker\n");
    exit (1);
  }
  pyramid->level= NULL;
  pyramid->nextlevel= NULL;

/*****
      level 1 linesegment determination
*****
*****/
/* sub0 = SUBSIZE; */          /* subimage size (may be fixed) */
printf("subsize(max 16) ==> "); /* subimage size (specify if desired) */
scanf("%d", &sub0);
curlevel = 1;                  /* current level number */
threshnum = sub0;              /* minimum number of points for a line */

/* set up subimage bounds and parameter space settings */
CenterImage(dx, dy);
SetSIbounds();
SetHTspace();

/* get level 1 line segments from the low level line extraction routine hhlow */
/* and set up the first level */
printf("subimages --> level 1 line segments0);
pyramid->level = hhlow(x, y, dx, dy);
pyramid->levelnum = curlevel;
pyramid->levelsub0 = sub0;
pyramid->levelrhodel = rhodel;
pyramid->levelthetadel = thetadel;
levrdel = pyramid->levelrhodel;
levtdel = pyramid->levelthetadel;
printf("level 1 done\n");

/*****
      grouping of line segments to yield longer lines (higher level)
*****
*****/
pyrlev = pyramid;          /* level 1 lines */
rhocells = 0;             /* initial value */
HTgrp[0] = NULL;
threshnum = GFSIZE;      /* minimum number of voters for a line group */

```

```

while (pyrlev != NULL) /* last level not empty */
{
  curlevel += 1; /* next level number */
  sub0 *= GSIZE; /* size of parent subimage */
  printf ("\nlevel %1d segments --> level %1d groups\n",
          pyrlev->levelnum, curlevel);
  SetHTspace(); /* set up HT space specifics */
  delrho = (int)((sqrt((double)2)/2)/rhodel) * rhodel;
  j = (int)(delrho/rhodel + .5) * 2 + 1;
  if (rhocells != j)
  {
    rhocells = j; /* rho cells for a theta row in voting array */
    if (HTgrp[0] != NULL)
      for (j=0; j<(1+2*GSIZE); j++)
        free(HTgrp[j]);

    /* set up HTaccu-like array for group vote accumulation */
    for (j=0; j<(1+2*GSIZE); j++)
    {
      HTgrp[j] = (acell *)malloc(rhocells * sizeof(acell));
      if (HTgrp[j] == NULL)
      {
        printf("malloc failed in HTgrp[%1d]\n", j);
        exit (1);
      }
    }
  }
}
SetSIbounds(); /* setup coordinate bounds of sibling subimages */
levhead = hhigh(pyramid, HTgrp); /* the new level */
if (levhead != NULL)
{
  /* get a new level marker */
  pyrlev = (levelmark *)malloc(sizeof(levelmark));
  if (pyrlev == NULL)
  {
    printf("malloc failed in pyrlev marker\n");
    exit (1);
  }
  pyrlev->level = levhead;
  pyrlev->nextlevel = pyramid; /* link the new level to last level */
  pyrlev->levelnum = curlevel; /* fill in new level specifications */
  pyrlev->levelsub0 = sub0;
  pyrlev->levelrhodel = rhodel;
  pyrlev->levelthetadel = thetadel;
  pyramid = pyrlev;
} /* endif new pyr level */
else
  pyrlev = NULL;
} /* endwhile more levels */
printf ("\npyramid complete\n");

/* release HTgrp space */
for (j=0; j<(1+2*GSIZE); j++)
  free(HTgrp[j]);

```

```

/*****
        postprocessing of the pyramid
*****/
printf("\npostprocessing pyramid\n");
hhpost(dx, dy, pyramid);

/* count lines and det. length of longest line in each level and */
/* generate output file of the lines */
SummarizeLevs(pyramid); /* generate a count-maxlen summary */
printf("\npostprocessing done\n");
printf("freeing pyramid\n");
pyramid = FreePyramid(pyramid);
printf("ALL DONE\n");
} /* end hough */

/*****/
/*****/
: Find the smallest exp2 square that may encompass image, the basis for a
: uniform subdivision of the image into an exp2 number of subimage windows.
: CALLED BY: hough
: INPUT  : image size -- dx columns, dy rows
*****/
void CenterImage(int dx, int dy)
{ int width; /* the bigger dimension */
  int wmod2; /* the exp2 square size */

/* center image in an exp2 square */
if (dx < dy) /* which is wider? */
    width = dy;
else
    width = dx;
wmod2 = 1;
while (wmod2 < width) /* find smallest exp2 square that fits image */
    wmod2 *= 2;
sb.xklead = (wmod2 - dx)/2; /* extra columns at each side of image */
sb.yklead = (wmod2 - dy)/2; /* extra rows at top and bottom of image */
} /* end CenterImage */

/*****/
: Set pointers to the first subimage bounds.
: CALLED BY: hough
*****/
void SetSIbounds()
{ int bstart, bend;

/* find first window containing a central subimage */
bstart = sb.yklead + sub0/2;
bend = 2 * sub0;
while ((bend - bstart) < (sub0/2))
    bstart = bstart - sub0;

```

```

sb.jstart = bstart;      /* first buffer row to contain an image row */
sb.jend = bend;         /* last buffer row to contain an image row */
sb.ykstart = 0 - bstart; /* start of subimage row in image coordinates */
sb.ykend = sb.ykstart + 2*sub0; /* end of subimage row in image coord */
sb.ykcentral = sb.ykstart + (sub0/2); /* start of central subimage */

bstart = sb.xklead + sub0/2;
bend = 2 * sub0;
while ((bend - bstart) < (sub0/2))
    bstart = bstart - sub0;
sb.istart = bstart;      /* first column: subimage window */
} /* end SetSIbounds */

/*****
: Set HT space limits and quantization intervals, and error limits
: CALLED BY : hough
*****/
void SetHTspace()
{
/* theta axis settings */
thetasize = 4 * sub0;      /* space size */
thetadel = (double)PI/thetasize; /* sampling interval */
deltheta = ((double)PI/sub0)/2; /* + - sampling error */

/* rho axis settings */
rhodel = (double)((int)((sub0 * sin(thetadel/2)) * 10 + 0.5))/10;
rhosize = (int)((sub0/2)/rhodel + 0.5) * 2 + 1;
rhozero = rhosize/2;      /* position: rho=0 */
} /* end SetHTspace */

/***** end hough.c *****/

```

APPENDIX F

THE PROGRAM SOURCE CODE FOR THE LOWEST LEVEL LINE

SEGMENT DETERMINATION ROUTINE OF HHOUGH


```

/*****
* hhlw.c
* Level 0 of the hierarchical approach to straight line extraction using
* the Hough Transform method.
* The binary thin-lined image is subdivided into subimages from which the
* line segments are extracted by the conventional Hough Transform
* method based on the normal equation form of straight lines.
* Line segments extracted comprise level 1 of the pyramid structure.
*
* CALLED BY : hhough
* INPUT : start of image location in image memory (x,y),
* image size: dx rows, dy columns.
* RETURNS : pointer to head of rowlists
*****/
#include "simp2.h"
#include "s2hhdefs.h"
#include <math.h>
#include <alloc.h>

/* from s2prim.c of SIMPP2 */
int read_hline(int x, int yk, int dx, PIXEL *simage);

/* internal functions */
void SetRows(int x, int y, int dx, int dy, PIXEL *simage[]);
void GetEdgePts(int dx, PIXEL *simage[], featpt *elist, int *ecnt, int *ccnt);
void ShuffleRows(int dy, PIXEL *simage[]);
void Parameterize(featpt *elist, int ecnt, float xs, float ys,
                 scpair SC[], int *HTaccu[]);
lineseg *FindPeaks(int *HTaccu[], float xs, float ys, lineseg *seg);
rowmark *GetRowMarker(void);

/* globals declared in hhough.c */
extern bounds sb; /* subimage bounds markers */
extern int sub0; /* subimage size */
extern rho_size, theta_size; /* rho-theta space size */
extern int rhozero; /* rho=0 position in rho-translated HTspace */
extern double rhodel, thetadel; /* rho-theta sampling intervals */
extern int threshnum; /* min feature points reqd for a true line */

/*****
*****/
rowmark *hhlw(int x, int y, int dx, int dy)
{ scpair LUTSC[SUBSIZE*4]; /* sin-cos look up table */
  PIXEL *simage[SUBSIZE*2]; /* subimage row buffer */
  int *HTaccu[SUBSIZE*3]; /* HT accumulator array */
  featpt *edlist, *hdlist; /* list of edge points found in subimage */
  int featcount, cencnt; /* number of edge points found: total, central */
  lineseg *oneseg, *headseg, *tailseg; /* line segment pointers */
  rowmark *levrow, *levhead, *levtail; /* pointers to rowmarkers */
  int j;

```

```

/* set up sin-cos look up table */
for (j=0; j<thetasize; j++)
{ LUTSC[j].sint = sin(thetadel * j);
  LUTSC[j].cost = cos(thetadel * j);
}

/* allocate HTaccu space */
for (j=0; j<rhosize; j++)
{ HTaccu[j] = (int *)malloc(thetasize*sizeof(int));
  if (HTaccu[j] == NULL)
  { printf("malloc failed in HTaccu[%1d] allocation\n", j);
    exit (1);
  }
}

/* allocate subimage row buffers */
for (j=0; j<(2*sub0); j++)
{ simage[j] = (PIXEL *)malloc(dx*sizeof(PIXEL));
  if (simage[j] == NULL)
  { printf("malloc failed in simage[%1d]\n", j);
    exit (1);
  }
}

/* allocate edge point list buffer */
hdlist = edlist = (featpt *)malloc(4*sub0*sub0*sizeof(featpt));
if (edlist == NULL)
{ printf("malloc failed in edlist allocation\n");
  exit (1);
}

/* initialize level 1 pointers */
levhead = levtail = levrow = NULL;

/* scan image for subimage line segments */
while (sb.ykcentral < dy) /* central subimage within dy */
{ SetRows(x, y, dx, dy, simage); /* take a subimage row */
  headseg = NULL; /* head of segment list for row */

/* scan the subimage row using subimage windows */
while (sb.xkcentral < dx) /* central subimage within dx */
{ GetEdgePts(dx, simage, edlist, &featcount, &cencount);
  edlist = hdlist;
/* find line segment/s if enough edgepoints */
if ((featcount >= threshnum) && (cencount > 0))
{ Parameterize(edlist, featcount, sb.xs, sb.ys, LUTSC, HTaccu);
  oneseq = FindPeaks(HTaccu, sb.xs, sb.ys, oneseq);
  if (oneseq != NULL) /* line segment/s found in the subimage */
  { if (headseg == NULL)
      headseg = oneseq; /* this is the first line segment in row */
    else

```

```

        tailseg->next = oneseq;
        while (oneseq->next != NULL) /* if more than one segment */
            oneseq = oneseq->next; /* find tail of linesegment list */
        tailseg = oneseq; /* remember the tail of the list */
    } /* endif oneseq */
} /* endif counts */
} /* endwhile xkcentral -- subimage along row */
if (headseg != NULL) /* there are line segments in row of subimages */
{ levrow = GetRowMarker(); /* get a rowmarker for the row of linesegs */
  levrow->row = headseg; /* hold on to the head of the row */
  levrow->ys = headseg->ys; /* note ys of the local subimage centers */
  if (levhead == NULL)
      levhead = levrow; /* it is the first row of line segments */
  else
      levtail->nextrow = levrow; /* link up new row to preceding row */
  levtail = levrow; /* last row so far formed in level */
}
ShuffleRows(dy, simage); /* set pointers for next row of subimages */
} /* endwhile ykcentral -- more rows to process */

/* all rows processed, release buffers */
hdlist = NULL;
free (edlist);
oneseq = headseg = tailseg = NULL;
levrow = levtail = NULL;
for (j=0; j<rhosize; j++)
    free(HTaccu[j]);
for (j=0; j<(2*sub0); j++)
    free(simage[j]);
return (levhead); /* return pointer to head of rowlists */
} /* end hflow */

/*****/
/*****/
| Set up the subimage buffer with the image rows for the subimage windows
| and set bounds for the first subimage window.
| CALLED BY : hflow
| INPUT : image location (x,y); image size (dx,dy);
| subimage row buffers, *simage[]
/*****/
void SetRows(int x, int y, int dx, int dy, PIXEL *simage[])
{ int yk; /* image row that will be stuffed into buffer */
  int i, j;

/* locate subimage ys center */
  sb.ys = (float)(sb.ykstart + (sb.ykstart + 2*sub0 - 1))/2;

/* locate yk -- image row to stuff into subimage buffer */
  if (sb.ykcentral < 0)
      yk = 0; /* row 1 of image to be stuffed into buffer */

```

```

else
    yk = sb.ykcentral + sub0/2; /* the first row that will get into buffer */

/* stuff subimage buffer from image array/memory */
if (yk < dy)
    for (j=sb.jstart, yk=y+yk; j<sb.jend; j++, yk++)
        read_hline(x, yk, dx, simage[j]); /* read a row of pixels */

/* set subimage buffer scan to start at first real image row */
if (sb.ykstart < 0)
    sb.ykstart = 0;          /* first row in image coordinates */
if ((sb.ykend - 2*sub0) < 0)
    sb.jstart = 2*sub0 - sb.ykend; /* first row in buffer coordinates */
else
    sb.jstart = 0;

/* start with first window containing central subimage */
sb.xkstart = 0 - sb.istart;
sb.xkend   = sb.xkstart + 2*sub0;
sb.xkcentral = sb.xkstart + sub0/2;
sb.iend    = sb.xkend;
} /* end SetRows */

/*****
| List edgepoints in the subimage window, count points within the central
| subimage, and adjust bounds for next window along the same subimage row.
| CALLED BY : hhlw
| INPUT   : image width, dx; subimage row buffers, *simage[];
|           edge list buffer, *elist; edge point count, *ecnt;
|           count of edge points found in central region, *ccnt
| *****/
void GetEdgePts(int dx, PIXEL *simage[], featpt *elist, int *ecnt, int *ccnt)
{ int xk, yk; /* pixel coordinates */
  featpt *head; /* pointer to head of edgelist */
  int xstartcent, xendcent; /* central subimage bounds, along x axis */
  int ystartcent, yendcent; /* central subimage bounds, along y axis */
  int i, j;

/* subimage xs center */
sb.xs = (float)(sb.xkstart + (sb.xkstart + 2*sub0 - 1))/2;

/* set first xk -- subimage pixel position */
if (sb.xkstart < 0)
    sb.xkstart = 0;
xk = sb.xkstart;
yk = sb.ykstart;

/* locate the bounding coordinates of the central subimage */
xstartcent = sb.xkcentral;
xendcent = xstartcent + sub0;
ystartcent = sb.ykcentral;

```

```

yendcent = ystartcent + sub0;

/* collect subimage edge points into edge list */
head = elist;
*ecnt = *ccnt = 0;
for (j=sb.jstart; j<sb.jend; j++, yk++) /* for each row of pixels */
{ for (i=sb.xkstart; i<sb.xkend; i++, xk++) /* scan along pixel row */
  if (simage[j][i] == MAXPIX) /* pixel is an edge pixel */
  { elist->xp = xk; /* note position into edgelist */
    elist->yp = yk;
    ++(*ecnt); /* count the edge pixel */
    /* check if edge pixel is in central subimage */
    if (((elist->xp >= xstartcent) && (elist->xp < xendcent))
        && ((elist->yp >= ystartcent) && (elist->yp < yendcent)))
      ++(*ccnt); /* count the edge pixel found in central region */
    ++elist;
  }
  xk = sb.xkstart; /* be ready for the next row of pixels */
}
head = NULL;

/* adjust bounds for next subimage along row */
sb.xkstart = sb.iend - sub0;
sb.xkcentral = sb.xkstart + (sub0/2);
sb.xkend = sb.xkstart + 2*sub0;
sb.iend = sb.xkend;
if (sb.xkend >= dx)
  sb.xkend = dx;
} /* end GetEdgePts */

/*****
| Shift subimage row buffer pointers, discarding the upper sub0 rows,
| advancing the lower sub0 rows, buffer made ready for another sub0 rows.
| CALLED BY : hhlw
| INPUT : number of image rows, dy; subimage row buffer, *simage[]
| *****/
void ShuffleRows(int dy, PIXEL *simage[])
{ PIXEL *temp; /* pointer to row of pixels */
  int j;

/* adjust bounds for next subimage row */
  if (sb.ykend != dy)
    sb.ykstart = sb.ykend - sub0;
  else
    sb.ykstart = sb.ykstart + sub0;
  sb.ykcentral = sb.ykstart + sub0/2;
  sb.ykend = sb.ykstart + 2*sub0;
  if (sb.ykend >= dy)
    sb.ykend = dy;
  sb.jstart = sub0;
  sb.jend = sb.ykend - sb.ykstart;

```

```

/* shuffle pointers; upper sub0 rows out for sub0 rows in */
if (sb.ykcentral < dy)
{ for (j=0; j<sub0; j++)
  { temp = simage[j];
    simage[j] = simage[j+sub0];
    simage[j+sub0] = temp;
  }
  temp = NULL;
}
} /* end ShuffleRows */

/*****
| Initialize HTaccu, Parameterize edge points in edge list and Accumulate
| counts for each rho-theta in the accumulator range.
| Parameterization: for each possible theta, rho is determined for each point
| Accumulation : each rho is adjusted for rhozero(for accumulator space)
| and when rho is within the rho-axis range, the corresponding rho-theta
| cell is incremented.
| CALLED BY : hllow
| INPUT : edge pixel list, *elist; edge pixel count, ecount;
| local subimage center (xs,ys); sin-cos look up table, SC[];
| HT accumulator array, *HTaccu[]
| *****/
void Parameterize(feapt *elist, int ecount, float xs, float ys,
                 scpair SC[], int *HTaccu[])
{ feapt *head; /* head of the edge list */
  double rho; /* rho parameter */
  int rspace; /* rho adjusted for accumulator space */
  int r, t;

/* initialize HTaccu */
for (r=0; r<rhosize; r++)
  for (t=0; t<thetasize; t++)
    HTaccu[r][t] = 0;

/* parameterize the edgelist */
head = elist;
for (t=0; t<thetasize; t++)
{ for (r=0; r<ecount; r++, elist++)
  { rho = (elist->xp - xs) * SC[t].cost + (ys - elist->yp) * SC[t].sint;
    if (rho < 0) /* adjust for accumulator space */
      rspace = (int)(rho/rhodel - 0.5) + rhozero;
    else
      rspace = (int)(rho/rhodel + 0.5) + rhozero;
    if ((rspace >= 0) && (rspace < rhosize)) /* edge pt in valid line */
      ++(HTaccu[rspace][t]);
  }
  elist = head;
}
head = NULL;
} /* end Parameterize */

```

```

/*****
| Scan the HTaccu for counts > threshnum.
| Each cell found denotes a valid line segment and is recorded as a linesegment
| node with the (ys,xs)-local subimage center and its (rho,theta) parameters.
| CALLED BY : hhlwo
| INPUT   : HT accumulator array, *HTaccu[]; subimage center (xs,ys);
|           line segment pointer, *seg.
| RETURNS : a list of line segments
*****/
lineseg *FindPeaks(int *HTaccu[], float xs, float ys, lineseg *seg)
{ lineseg *oneline; /* line segment found */
  int r, t;

  seg = NULL;
  /* scan HTaccu for counts >= threshnum */
  for (r=0; r<rhosize; r++)
    for (t=0; t<thetasize; t++)
      if (HTaccu[r][t] >= threshnum)
        { oneline = (lineseg *)malloc(sizeof(lineseg));
          if (oneline == NULL)
            { printf("malloc failed in FindPeak lineseg allocation\n");
              exit (1);
            }
          /* record the line segment's parameters and subimage center */
          /* store -1 in len field, will be used later in the higher grouping */
          oneline->ys = ys;
          oneline->xs = xs;
          oneline->thetas = t;
          oneline->rhos = r - rhozero;
          oneline->len = -1;
          oneline->subseg = NULL; /* no subsegments yet */
          oneline->next = seg; /* link to list of line segs found in subimage*/
          seg = oneline;
        } /* endif >= threshnum */
  oneline = NULL;
  return (seg); /* return the head of list of line segments found */
} /* end FindPeaks */

/*****
| Get a row marker for the list of line segments found in a row of subimages.
*****/
rowmark *GetRowMarker()
{ rowmark *rowmarker;

  rowmarker = (rowmark *)malloc(sizeof(rowmark));
  if (rowmarker == NULL)
    { printf("malloc failed in rowmarker allocation.\n");
      exit (1);
    }
  rowmarker->row = NULL;
  rowmarker->nextrow = NULL;
}

```

```
    return (rowmarker);  
} /* end GetRowMarker */
```

```
/****** end hflow.c *****/
```


APPENDIX G

THE PROGRAM SOURCE CODE FOR THE HIGHER LEVEL
LINE SEGMENT GROUPING ROUTINE OF HHOUGH

```

/*****
* hhigh.c
* higher level grouping for the hierarchical HT method
* A neighborhood of 4x4 subimages is checked for line segments that are
* collinear to the line segments found in the central 2x2 subimage region.
* CALLED BY : hhough
* INPUT : the last level, *pyramid; HT accumulator cells, *HTgrp[].
*****/
#include "simpp2.h"
#include "s2hhdefs.h"
#include <math.h>
#include <alloc.h>

/* internal functions */
rowmark *SetSegRows(rowmark *currow);
void SetSegWindow(int *fcount, int *ccount);
void LabelCentral(int cstart, int cend);
void GetRpar(double tp, float xp, float yp, lineseg *fpt,acell *HTrow,int sw);
void GetVotes(int j, int i, lineseg *fpt, float yp, float xp, double Trange[],
              acell *HTgrp[]);
lineseg *PeakGrp(int fj, int fi, double Trange[], acell *HTgrp[],int *ccount);
featpt *GetSubseg(lineseg *fpt, featpt *headseg);
void FinalLabels(int start, int end);
void CleanRows(int end);
void ShuffleSegRows(void);
void ShuffleSegWindow(void);
void CleanLevel(levelmark *lastlevel);

/* in hhlow.c */
rowmark *GetRowMarker(void);

/* globals declared in hhough.c */
extern int sub0;          /* subimage size */
extern bounds sb;       /* subimage bounds markers */
extern int rhosize;     /* rho space size */
extern int rhozero;     /* rho=0 position */
extern double rhodel, thetadel; /* rho-theta sampling intervals */
extern double delrho, deltheta; /* rho-theta error limits */
extern int threshnum;   /* min feat pts for true line */
extern int curlevel;   /* current pyramid level */
extern int rhocells;   /* width of voting list (grouping) */
extern double levrdel, levtdel; /* rhodel,thetadel values for last level */

rowmark *segrow[GFSIZE*2]; /* ptrs to rows of lower level linesegs */
lineseg *segmark[GFSIZE*2]; /* ptrs to linesegs last considered */
lineseg *window[GFSIZE*2][GFSIZE*2]; /* linesegs in sibling subimages for grpg*/

/*****
*****/
rowmark *hhigh(levelmark *pyramid, acell *HTgrp[])
{ int fptcount, cccount; /* number of featpts in current window */

```

```

double Trange[GSIZE*2 + 1]; /* range of possible theta values */
lineseg *oneseg, *headseg, *tailseg, *fpt; /* line segment pointers */
rowmark *levrow, *levhead, *levtail; /* row pointers for new level */
rowmark *prevlev; /* pointer to a row in the last level */
int j, i;

/* initialize the new level pointers */
levhead = levtail = levrow = NULL;

/* the last level */
prevlev = pyramid->level;
levrdel = pyramid->levelrhodel;
levtdel = pyramid->levelthetadel;

/* form the new level */
sb.ykcentral = GSIZE; /* indicator of central group presence */
while ((prevlev != NULL) || (sb.ykcentral >= (int)GSIZE/2))
/* take a set of lower level rows of segments */
{ prevlev = SetSegRows(prevlev);
  headseg = NULL; /* head of line segment list for row */
  while ((sb.jend > 0) || (sb.xkcentral > 0))
  { SetSegWindow(&fptcount, &cencount);
    if ((fptcount >= threshnum) && (cencount > 0))
    { LabelCentral(GSIZE/2, GSIZE+GSIZE/2);
      j = GSIZE/2;
      /* take a central feature point at a time */
      while (j < (GSIZE+GSIZE/2))
      { i = GSIZE/2;
        while (i < (GSIZE+GSIZE/2))
        { if (window[j][i] != NULL)
          { fpt = window[j][i];
            /* get voters for each featpt in the subimage */
            while ((fpt != NULL) && (fpt->xs < (window[j][i]->xs +.5)))
            { if (fpt->len > -2)
              { GetVotes(j,i,fpt, sb.ys, sb.xs, Trange, HTgrp);
                /* form a line segment from highest voted group */
                oneseg = PeakGrp(j,i, Trange, HTgrp, &cencount);
                if (oneseg == NULL)
                { fpt->len = -3; /* not group central feat pt */
                  cencount--;
                }
              }
            else /* a group was found, form a line segment */
            { oneseg->xs = sb.xs;
              oneseg->ys = sb.ys;
              oneseg->len = -1;
              if (headseg == NULL) /* first line seg in row */
                headseg = oneseg;
              else /* link new line to tail of list */
                tailseg->next = oneseg;
              tailseg = oneseg; /* hold on to the tail */
            }
          }
        }
      }
    }
  }
} /* endif oneseg */

```

```

    } /* endif fpt->len > -2 : not voted */
    if (cencount > 0)
        fpt = fpt->next; /* get the next central feat pt */
    else /* all central feature points done */
        { j = 2*GSIZE;
          i = 2*GSIZE;
          break;
        }
    } /* endwhile same subwindow */
} /* endif window != NULL */
i++;
} /* endwhile i */
j++;
} /* endwhile j */
FinalLabels(GSIZE/2, GSIZE+GSIZE/2);
} /* endif group segments */
ShuffleSegWindow(); /* set pointers to next set of sibling subimages*/
} /* endwhile within row */
if (headseg != NULL)
{ levrow = GetRowMarker(); /* for new line segment groups in the row */
  levrow->row = headseg;
  levrow->ys = headseg->ys;
  if (levhead == NULL)
      levhead = levrow; /* first rowlist */
  else
      levtail->nextrow = levrow; /* link up to last rowlist formed */
  levtail = levrow;
} /* endif levrow */
CleanRows(GSIZE); /* delete grouped segments from lower level */
ShuffleSegRows(); /* get another set of rows from lower level */
} /* endwhile more rows */
CleanRows(2*GSIZE); /* delete last grouped segs from lowlevel */
if (levhead != NULL)
    CleanLevel(pyramid); /* get rid of rowmarkers that carry no line-segs */
oneseq = headseg = tailseg = fpt = NULL;
levrow = levtail = prevlev = NULL;
return (levhead);
} /* end hhigh */

```

```

/*****
/*****
| set up pointers to headsegments of rows for the overlapped subimage group
| CALLED BY : hhigh
| INPUT   : the row to be processed from the last level
| RETURNS : the row that will be taken first in the next round
*****/
rowmark *SetSegRows(rowmark *currow)
{ rowmark *subrow; /* pointer to row of last level */
  int yk; /* note position of row in terms of image space coordinates */
  int jstart; /* the last level's row that will become a part of the subimm */

```

```

int j;

/* find ys of subimage center */
sb.ys =(float)(sb.ykstart + sb.ykend -1)/2;
yk = sb.ykstart + sub0/2 -1;

/* determine the y position of the row in terms of image space coordinates */
if (sb.jend == 2*sub0)
    sb.jstart = 0;
else
    yk = yk + sub0;

sb.jend = 0;      /* indicative of non-empty window pointers */
/* set pointers to rows of the last level that will contain the sibling */
/* subimages of the new level's row */
subrow = currow;
for (j=sb.jstart; yk < sb.ykend; yk += sub0/2, j++)
    if ((subrow != NULL) && (subrow->ys < yk))
        { segrow[j] = subrow;
          sb.jend++;
          subrow = subrow->nextrow;
        }
    else
        segrow[j] = NULL;

/* set up lineseg markers along the rows */
for (j=0; j<(2*GSIZE); j++)
    if (segrow[j] != NULL)
        segmark[j] = segrow[j]->row;
    else
        segmark[j] = NULL;

/* set up first window bounds along row */
sb.xkstart = 0 - sb.istart;
sb.xkend  = sb.xkstart + 2*sub0;
sb.iend = 0;      /* start of window column to be filled up */
sb.xkcentral = GSIZE; /* indicator of number in central window */

return(subrow);
} /* end SetSegRows */

/*****
| set row bounds for next subimage row, shuffle pointers to displace upper
| half rows, preparing lower half pointers for the next "group" rows.
| CALLED BY : hhigh
*****/
void ShuffleSegRows()
{ int j;

/* make ready for next set of rows for the windows */
sb.ykstart = sb.ykstart + sub0;

```

```

sb.ykend = sb.ykstart + 2*sub0;

/* shuffle rowmark pointers: displace upper "group" number of rows */
sb.ykcentral = 0;
for (j=0; j<GSIZE; j++)
{ segrow[j] = segrow[j+GSIZE];
  if (segrow[j] != NULL)
    sb.ykcentral = j;
}
sb.jstart = GSIZE; /* where to start adding more rows from lower level */
} /* end ShuffleSegRows */

/*****
| set up the pointers to line segments comprising a window along the rows for
| the overlapped subimage group
| CALLED BY : hhigh
| INPUT    : feature points count in the overlapped region, *fcount;
|           feature points count in the central subimage region, *ccount.
| *****/
void SetSegWindow(int *fcount, int *ccount)
{ int cstart, cend; /* central subimage */
  int count;      /* feat pt count */
  int xstart, xk; /* x positions in terms of image space coordinates */
  lineseg *temp; /* pointer to line segments(feature points) */
  int i, j;

  /* find center of new subimage */
  sb.xs = (float)(sb.xkstart + sb.xkend - 1)/2;

  /* start of the overlapped subimage window */
  xstart = sb.xkstart + sub0/2 - 1;
  if (sb.iend == GSIZE)
    xstart = xstart + sub0;

  *fcount = *ccount = 0;
  cstart = GSIZE/2;
  cend = cstart + GSIZE;
  /* set up pointers to feature points in the overlapped subimage window */
  for (j=0; j<(2*GSIZE); j++)
  { if (segmark[j] == NULL)
    { for (i=sb.iend; i<(2*GSIZE); i++)
      window[j][i] = NULL;
    }
    else
    { for (xk=xstart, i=sb.iend; xk < sb.xkend; xk += sub0/2, i++)
      { if ((segmark[j] != NULL) && (segmark[j]->xs < xk))
        { window[j][i] = segmark[j];
          count = 0;
          while ((segmark[j] != NULL) && (segmark[j]->xs < xk))
            { count++;
              segmark[j] = segmark[j]->next;
            }
        }
      }
    }
  }
}

```

```

    }
    *fcount += count;
    if ((i>=cstart) && (i<cend) && (j>=cstart) && (j<cend))
        *ccount += count;
    }
    else
        window[j][i] = NULL;
    }
} /* endif segmark */
} /* endfor j */

/* count feat points in old window columns */
if (sb.iend == GSIZE)
    for (j=0; j<(2*GSIZE); j++)
        for (i=0; i<GSIZE; i++)
            if (window[j][i] != NULL)
                { temp = window[j][i];
                  count = 0;
                  while ((temp != NULL) && (temp->xs < (window[j][i]->xs +.5)))
                      { count++;
                        temp = temp->next;
                      }
                  *fcount += count;
                  if ((i>=cstart) && (i<cend) && (j>=cstart) && (j<cend))
                      *ccount += count;
                }
    temp = NULL;
} /* end SetSegWindow */

/*****
| set bounds for next segment window, and shuffle the window pointers to
| displace the left half segment columns, the right half to be filled in anew.
| CALLED BY : hhigh
*****/
void ShuffleSegWindow()
{ int j, i;

/* set up bounds for next window */
sb.xkstart = sb.xkstart + sub0;
sb.xkend = sb.xkstart + 2*sub0;
sb.iend = GSIZE; /* start of window columns to be filled */

/* check if there are more line segments left to shift into the window */
sb.jend = 0;
for (j=0; j<(2*GSIZE); j++)
    if (segmark[j] != NULL)
        sb.jend++;

/* check if there would be a central subimage part left after shuffle */
sb.xkcentral = 0;
for (j=(GSIZE/2); j<(GSIZE+GSIZE/2); j++)

```

```

    for (i=(GSIZE+GSIZE/2); i<(2*GSIZE); i++)
        if (window[j][i] != NULL)
            sb.xkcentral++;

/* shuffle segment window pointers: displace left "group" number of columns */
if ((sb.jend > 0) || (sb.xkcentral > 0))
{ for (j=0; j<(2*GSIZE); j++)
  for (i=0; i<GSIZE; i++)
    window[j][i] = window[j][i+GSIZE];
}
} /* end ShuffleSegWindow */

/*****
| Label central feature points (using the len field) with ints > 0.
| to differentiate the featpts from the featpts in the supporting siblings.
| (len field of featpts in supporting siblings may be:
| -1 : not processed yet if in a later subimage,
| or not member of any earlier groups if in an earlier subimage,
| 0 : already a group member in earlier groupings.
| CALLED BY : hhigh
| INPUT : bounds of the central subimage region, cstart and cend.
*****/
void LabelCentral(int cstart, int cend)
{ int i, j, label;
  lineseg *temp;

  label = 1;
  for (j=cstart; j<cend; j++)
    for (i=cstart; i<cend; i++)
      if (window[j][i] != NULL)
        { temp = window[j][i];
          while ((temp != NULL) && (temp->xs < (window[j][i]->xs +.5)))
            { temp->len = label++;
              temp = temp->next;
            }
        }
  temp = NULL;
} /* LabelCentral */

/*****
| Finalize flags(field len) of central feature points:
| -3 --> -1 : no group membership
| -2 --> 0 : a group member
| CALLED BY : hhigh
| INPUT : bounds of the central subimage region, start and end.
*****/
void FinalLabels(int start, int end)
{ int j, i;
  lineseg *fpt; /* a line segment (feature point) */

  for (j=start; j<end; j++)

```



```

    for (i=start; i<end; i++)
        if (window[j][i] != NULL)
            { fpt = window[j][i];
              while ((fpt != NULL) && (fpt->xs < (window[j][i]->xs +.5)))
                  { fpt->len += 2;
                    fpt = fpt->next;
                  }
            }
    fpt = NULL;
} /* end FinalLabels */

/*****
| For each central feature point:
| initialize voting arrays and find support for the feature point from its
| immediate neighbors.
| flag(len field): support area=> 0:grpmember, -1:nogrp
|                   central area=> >0:not processed,
|                   -2:processed/grpmember,
|                   -3:processed/nogrp
| CALLED BY : hhigh
| INPUT   : the feature point seeking a group, *fpt;
|           position of the feature point in window (j,i);
|           the feature point's lower level subimage center (yp, xp);
|           range of theta allowed by the feature point's collinearity
|           constraint, Trange[]; the voting array, *HTgrp[]
| *****/
void GetVotes(int j, int i, lineseg *fpt, float yp, float xp,
              double Trange[], acell *HTgrp[])
{ double fmin, fmax, tp; /* the featpt's theta-range */
  double nmin, nmax; /* the neighbor's theta-range */
  lineseg *npt; /* the neighbor */
  int nj, ni; /* window coordinates of the immediate neighbors */
  int k;

/* initialize Trange markers */
  for (k=0; k<(GFSIZE*2+1); k++)
      Trange[k] = PI+ thetadel/2;

/* find groups for each central feature point */
  fmin = (fpt->thetas * levtdel) - deltheta;
  fmax = (fpt->thetas * levtdel) + deltheta;
  if (fmin < 0)
      fmin = 0;
  if (fmax > (PI - thetadel/2))
      fmax = PI - thetadel;

/* set up theta range values and get rho parameters of the featpt */
  for (tp=fmin, k=0; tp < (fmax + thetadel/2); tp += thetadel, k++)
      { Trange[k] = tp;
        GetRpar(tp, xp, yp, fpt, HTgrp[k], 0);
      }
}

```

```

/* examine immediate neighbors */
for (nj=(j-1); nj<=(j+1); nj++)
  for (ni=(i-1); ni<=(i+1); ni++)
    if ((window[nj][ni] != NULL) && (window[nj][ni] != window[j][i]))
      { npt = window[nj][ni];
        while ((npt != NULL) && (npt->xs < (window[nj][ni]->xs +.5)))
          { if (npt->len > (-2)) /* central:notdone*/
              { /* get neighbor's theta range */
                nmin = (npt->thetas *levtdel) - deltheta;
                nmax = (npt->thetas *levtdel) + deltheta;
                /* check for overlap with the feature point's theta range */
                if (((nmin + thetadel/2) > fmin)&&((nmin - thetadel/2) < fmax))
                  nmax = fmax;
                else if(((nmax-thetadel/2)<fmax) && ((nmax+thetadel/2)>fmin))
                  nmin = fmin;
                else
                  nmin = nmax = PI + thetadel/2;
                if (nmin < (PI- thetadel/2))
                  { for(tp=fmin,k=0; tp<(nmin-thetadel/2); tp +=thetadel,k++);
                    for(tp=Trange[k]; tp<(nmax+thetadel/2); tp +=thetadel,k++)
                      { /* theta ranges overlap, get neighbor's vote */
                        GetRpar(Trange[k], xp, yp, npt, HTgrp[k], 1);
                      }
                    } /* endif nmin != PI */
                } /* endif npt->len > -2 */
            npt = npt->next; /* next featpt within window */
          } /* endwhile within window */
        } /* endif npt */
    } /* endfor ni */
  } /* endfor nj */
npt = NULL;
} /* end GetVotes */

```

```

/*****
| Get the rho parameters for the point and tab appropriate cell in HTgrp.
| (a vote is added in the H->vote, the featpt label recorded in H->voter[]
| and H->voter[0] incremented for each central featpt voter)
| note: sw=0 for the featpt getting votes, sw=1 for voting neighbors.
| CALLED BY : GetVotes
| INPUT : fpt's lower level subimage center (xp,yp); theta value to be
| sampled, tp; row of the voting array corresponding to tp, *HTrow
| type of feature point being parameterized, sw.
| *****/

```

```

void GetRpar(double tp, float xp, float yp, lineseg *fpt, acell *HTrow,int sw)
{ double rs, rp; /* rho of sibling subimage, rho of parent */
  double frho, fheta; /* param values of sibling in its level */
  int rspace, lastrspace; /* rho in quantized parameter space */
  acell *H; /* an accumulator cell */
  int i, r, m;

```

```

/* initialize the voting array */

```

```

H = HTrow;
if (sw == 0)
{ for (i=0, H=HTrow; i<rhocells; i++, H++)
  { H->vote = 0;
    H->rhos = -1;
    H->voter[0] = 0; /* number of voters for the cell */
  }
  H = HTrow;
  m = 0;
}

/* adjust rho from lower level subimage to the new level subimage */
lastrspace = rhosize; /* to watch out for possible duplicated votes */
frho = fpt->rhos * levrdel;
ftheta = fpt->thetas * levtdel;
for(rs=(frho - delrho),r=0; rs<(frho + delrho + rhodel/2); rs += rhodel,r++)
{ rp = rs * cos(tp - ftheta)
  - (xp - fpt->xs)*cos(tp) - (fpt->ys - yp)*sin(tp);
  if (rp < 0) /* rho in parent subimage */
    rspace = (int)(rp/rhodel - .5) + rhozero;
  else
    rspace = (int)(rp/rhodel + .5) + rhozero;

  /* rspace must be within range and must not be a duplication */
  if ((rspace < 0) || (rspace >= rhosize) || (rspace == lastrspace))
    i = rhocells;
  else
  { lastrspace = rspace;
    if (sw == 0) /* initialization of voting array params */
      i = m;
    else /* sw == 1 : a neighbor voter */
      for (i=0,H=HTrow; ((i<rhocells)&&(rspace != H->rhos)); i++,H++);
  } /* endif rspace within space range */

  if (i < rhocells) /* may vote */
  { if (sw == 0)
    H->rhos = rspace;
    if (r == (rhocells-1)/2) /* vote weight is 1 for rs=rho of fpt */
      H->vote += 1;
    else
      H->vote += 0.99;
    if (fpt->len > 0) /* is a central fpt voter */
    { H->voter[0] += 1; /* number of central fpt voters so far */
      H->voter[H->voter[0]] = fpt->len;
    }
    if (sw == 0)
    { m++;
      H++;
    }
  } /* endif may vote */
} /* endfor rho range of sibling */

```

```

    H = NULL;
} /* end GetRpar */

/*****
| find maximum voted cell > 1, make a lineseg,
| change flags(len field) of grouped central featpts (label --> -2),
| adjust fptcount, cencount.
| CALLED BY : hhigh
| INPUT   : the feature points position in the window (fj, fi);
|           theta range of feature point Trange[]; the voting array *HTgrp[];
|           count of central feature points, *ccount.
| RETURNS : a new line segment (a group)
*****/
lineseg *PeakGrp(int fj, int fi, double Trange[], acell *HTgrp[], int *ccount)
{ float vmax;          /* highest vote */
  int imax, jmax;     /* i,j of highest voted cell */
  int voters;        /* num of cent voters */
  acell *H;          /* pointer to voting array cells */
  featpt *member, *lastmember; /*center of lowest level subim contg segment*/
  lineseg *fpt, *onegrp; /* new group */
  int j, i, k;

/* search for the highest vote */
  vmax = 1;
  for (j=0; ((j<(GFSIZE*2+1)) && (Trange[j] < PI)); j++)
    for (i=0, H=HTgrp[j]; ((i<rhocells) && (H->vote > 0)); i++, H++)
      if (H->vote > vmax)
        { vmax = H->vote;
          imax = i;
          jmax = j;
        }

/* create group if maximum found > 1 */
  onegrp = NULL;
  if (vmax > 1) /* group found */
    { voters = HTgrp[jmax][imax].voter[0];
      onegrp = (lineseg *)malloc(sizeof(lineseg));
      if (onegrp == NULL)
        { printf("malloc failed in onegrp\n");
          exit (1);
        }
      onegrp->rhos = HTgrp[jmax][imax].rhos - rhozero;
      onegrp->thetas = (int)(Trange[jmax] /thetadel +.5);
      onegrp->next = NULL;
      onegrp->subseg = NULL;
      lastmember = NULL;

/* connect central featpt voters' subsegments to new group */
      j = fj - 1;
      while (j<(fj+2))
        { i = fi - 1;

```

```

while (i<(fi+2))
{ if (window[j][i] != NULL)
  { fpt = window[j][i];
    while ((fpt != NULL) && (fpt->xs < (window[j][i]->xs +.5)))
    { k = 1;
      while (k < (voters + 1))
      { /* check if fpt is one of the voters */
        if (fpt->len == HTgrp[jmax][imax].voter[k]) /* voter */
        { if (curlevel < 3) /* lineseg has no members */
          { /* has to create a subsegment node first */
            member = GetSubseg(fpt, onegrp->subseg);
            if (onegrp->subseg == NULL)
            { onegrp->subseg = member; /*first subseg in list*/
              lastmember = member;
            }
            else if (member != NULL) /*link up to subseg list*/
            { lastmember->next = member;
              lastmember = member;
            }
          }
        }
        else /* lineseg has a string of members */
        { member = fpt->subseg;
          fpt->subseg = NULL;
          if (onegrp->subseg == NULL)
            onegrp->subseg = member;
          else
            lastmember->next = member;
          while (member->next != NULL)
            member = member->next;
          lastmember = member;
        } /* endif curlevel < 3 */
        fpt->len = -2; /* a grouped central featpt */
        HTgrp[jmax][imax].voter[0] -= 1;
        *ccount -= 1;
        break;
      } /* endif fpt voted */
      else
        k += 1; /* check next voter number */
    } /* endwhile central voter list */
    if (HTgrp[jmax][imax].voter[0] > 0)
      fpt = fpt->next;
    else /* voters' list empty, end of group formation */
    { i = fi + 2;
      j = fj + 2;
      break;
    }
  } /* endwhile same window */
} /* endif fpt */
i++;
} /* endwhile i */
j++;

```

```

    } /* endwhile j */
    fpt = NULL;
    member = lastmember = NULL;
} /* endif group found */
H = NULL;
return (onegrp);
} /* end PeakGrp */

/*****
| Get member linesegments (in terms of level 0 subimage centers) for new group.
| CALLED BY : PeakGrp
| INPUT   : the feature point, *fpt; the new group's subsegment list *headseg
| RETURNS : a subsegment node containing the member's lowest level subimage
|           center coordinates.
| *****/
featpt *GetSubseg(lineseg *fpt, featpt *headseg)
{ featpt *mhd; /* temporary pointer to *headseg */
  featpt *newseg; /* the subsegment node created for the member line segment */

  mhd = headseg;
  newseg = NULL;

  /*is seg already member?, a segment must get included just once */
  while (mhd != NULL)
    if ((mhd->xp == fpt->xs)&&(mhd->yp == fpt->ys))
      break;
    else
      mhd = mhd->next;

  if (mhd == NULL) /* a new member */
  { newseg = (featpt *)malloc(sizeof(featpt));
    if (newseg == NULL)
    { printf ("malloc failed in newseg\n");
      exit (1);
    }
    newseg->xp = fpt->xs;
    newseg->yp = fpt->ys;
    newseg->next = NULL;
  }
  mhd = NULL;
  return (newseg);
} /* end GetSubseg */

/*****
| delete segments in rows to be displaced (segrows<group or the last segrows)
| that have become group members (len-->0) in the higher level
| CALLED BY : hhigh
| INPUT   : number of rows that must be cleaned up, end -- corresponds to
|           the upper two rows of the lower level that will be shifted out
|           of the overlapped subimage rows.
| *****/

```

```

void CleanRows(int end)
{ lineseg *test, *last; /* ptrs to line segments in row being cleaned up */
  int j;

  for (j=0; j<end; j++)
  { if (segrow[j] != NULL)
    { test = segrow[j]->row;
      while ((test != NULL) && (test->len == 0))
        { segrow[j]->row = test->next;
          test->next = NULL;
          free(test);
          test = segrow[j]->row;
        }
      if (test != NULL)
        { last = test;
          test = test->next;
          while (test != NULL)
            if (test->len == 0)
              { last->next = test->next;
                test->next = NULL;
                free(test);
                test = last->next;
              }
            else
              { last = test;
                test = test->next;
              }
          /* endwhile test != NULL */
        } /* endif test=segrow[j]->row != NULL */
    } /* endif segrow[j] != NULL */
  } /* endfor segrow[j] < group */
} /* end CleanRows */

/*****
| delete rowmarks in the previous level that no longer support row segments.
| (rowmarks->row == NULL)
| CALLED BY : hhigh
| INPUT   : the last level
|******/
void CleanLevel(levelmark *lastlevel)
{ rowmark *test, *last; /*ptrs to rowmarks that will be checked for linesegs*/

  test = lastlevel->level;
  while ((test != NULL) && (test->row == NULL))
  { lastlevel->level = test->nextrow;
    test->nextrow = NULL;
    free(test);
    test = lastlevel->level;
  }
  if (test != NULL)
  { last = test;

```

```
test = test->nextrow;
while (test != NULL)
  if (test->row == NULL)
    { last->nextrow = test->nextrow;
      test->nextrow = NULL;
      free(test);
      test = last->nextrow;
    }
  else
    { last = test;
      test = test->nextrow;
    }
} /* endif test != NULL */
} /* end CleanLevel */
```

```
/****** end hhigh.c *****/
```


APPENDIX H

THE PROGRAM SOURCE CODE FOR THE PRINTING
AND REMAPPING ROUTINES FOR LINES
EXTRACTED BY HHOUGH

```

/*****
* hhout.c
* Printout and remapping routines for the lineseg pyramid formed by the
* hierarchical approach to the HT method
* Contains:
*   RemapLevel() : draws the lines found in the level in a specified
*   quadrant of image memory and displays the resulting image of
*   lines on the external monitor;
*   WritePyrPspace() : writes the description of each line at each
*   level of the postprocessed hierarchy in a file.
*   The description is in terms of the line's midpoint,
*   its (rho,theta) parameters, its length and the rectangular
*   dimensions (delta x, delta y) of the area that encompasses the
*   line if it is neither horizontal nor vertical;
*   PrintPyramid() : writes the line descriptions at each level of the
*   hierarchy to stdout. The description consists of the
*   (rho, theta) parameters of the line, the line's subimage center,
*   the label stored in the len field of the lineseg node, and
*   a list of the subsegments if the hierarchy has not been
*   postprocessed yet, otherwise, the output is similar to that of
*   WritePyrPspace;
*   PrintLevel() : writes the descriptions of the lines found in a level
*   of the hierarchy to stdout;
*   PrintRowSegs() : writes the descriptions of the lines found in a row
*   of subimages to stdout.
*   PrintOneSeg() : writes the description of a single line segment to
*   stdout.
*
* Calls to these routines may be inserted at strategic points in
* hthough, hhlow, hhhigh or hhpost to monitor the lines as desired.
*****/
#include "simpp2.h"
#include "s2hhdefs.h"
#include <math.h>
#include <alloc.h>

/* in hhpost.c */
void GetEnds(lineseg *fpt, int hsize, int dx, int dy);
void CheckLimits(int dx, int dy);
void DeriveName(char *ext);

/* from s2inter.c of SIMPP2 */
void write_pixel(int x, int y, PIXEL z);

/* globals declared in hthough.c */
extern bounds sb; /* subimage bounds markers */
extern double levrdel,levtdel; /*lev's rhodel, thetadel for accuvals->parvals*/

```

```

/* input image filename to be used to derive outfile name */
/* declared in the driver program for the complete low to */
/* medium level image processing system or in the driver */
/* program for testing hthough on line-thinned images */
extern char name[];

/* vars common to hhpost, ReMapLevel */
/* declared in hhpost.c */
extern double halftdel; /* half of the level's thetadel value */
extern int halfsize; /* half of the level's subimage size */
extern double frho, ftheta; /* rho, theta values in parameter space */
extern int xmin, xmax, ymin, ymax; /* bounds for area cont'g the lineseg */

/*****
| remap level
| remap linesegs found in the level in a quadrant of the image memory.
| INPUT : the starting point (x,y) of the quadrant of width dx and height dy;
| the level whose lines are to be remapped, *curlev; the current
| state of the hierarchy, swP, i.e., swP = 0, while the hierarchy
| is still being developed, swP = 1, when the hierarchy has been
| postprocessed already.
*****/
void ReMapLevel(int x,int y,int dx,int dy,levelmark *curlev,int swP)
{ int xi, yi; /* the coordinates of the point image space */
  float xp, yp; /* the coordinates of point wrt subimage center */
  rowmark *currow; /* the current row of line segments */
  lineseg *fpt; /* a feature point */

  /* retrieve level's accumulator array parameters */
  if (swP == 0)
  { halftdel = curlev->levelthetadel /2;
    halfsize = curlev->levelsub0 /2;
    for (xi = (curlev->levelnum -1); xi > 0; xi--)
      halfsize = halfsize/2;
  }
  levrdel = curlev->levelrhodel;
  levtdel = curlev->levelthetadel;

  /* process each line segment for remapping */
  printf(":::level %ld0, curlev->levelnum);
  /* for every row of line segments */
  for (currow = curlev->level; currow != NULL; currow = currow->nextrow)
    /* for every feature point along row */
    for (fpt = currow->row; fpt != NULL; fpt = fpt->next)
    { /* convert the rho, theta in accumulator coordinates */
      /* to image space coordinates */
      frho = fpt->rhos * levrdel;
      ftheta = fpt->thetas * levtdel;
      if (swP == 0)
        GetEnds(fpt, halfsize, dx, dy);
    }
}

```

```

/***** remap the lineseg *****/
if (ftheta < halftdel) /* vertical line */
{ /* VERTICAL LINE */
  if (swP == 0)
  { xp = fpt->xs + frho;
    if (xp < 0)
      xi = (int)(xp - 0.5);
    else
      xi = (int)(xp + 0.5);
  }
  else
  { xi = fpt->xs;
    xmin = xi;
    ymin = (int)(fpt->ys - ((float)fpt->len/2) + 1);
    ymax = ymin + fpt->len;
    xmax = xi + 1;
    /* ensure limits are within the image space */
    CheckLimits(dx, dy);
  }
  if ((xi >= xmin) && (xi < xmax))
  { for (yi=ymin; yi<ymax; yi++)
      write_pixel(x+xi, y+yi, MAXPIX);
  }
  }
  else if ((ftheta > (PI/2 - halftdel))
    && (ftheta < (PI/2 + halftdel)))
  { /* HORIZONTAL LINE */
  if (swP == 0)
  { yp = fpt->ys - frho;
    if (yp < 0)
      yi = (int)(yp - 0.5);
    else
      yi = (int)(yp + 0.5);
  }
  else
  { yi = fpt->ys;
    xmin = (int)(fpt->xs - ((float)fpt->len/2) + 1);
    xmax = xmin + fpt->len;
    ymin = yi;
    ymax = yi + 1;
    CheckLimits(dx, dy);
  }
  if ((yi >= ymin) && (yi < ymax))
  { for (xi=xmin; xi<xmax; xi++)
      write_pixel(x+xi, y+yi, MAXPIX);
  }
  }
} /* endelseif horizontal */
else /* SLANTING LINE */
{ if (swP == 1)
  { xp = fpt->subseg->xp;
    yp = fpt->subseg->yp;

```

```

if (xp < 0)
    xp = -1 * xp;
xmin = (int)(fpt->xs - xp/2);
xmax = (int)(xmin + xp + 1);
if (yp < 0)
    yp = -1 * yp;
ymin = (int)(fpt->ys - yp/2);
ymax = (int)(ymin + yp + 1);
CheckLimits(dx, dy);
}
if ((ftheta > (PI/4 - halftdel))
    && (ftheta < (3*PI/4 + halftdel)))
/* |slope| <= 1 */
for (xi=xmin; xi<xmax; xi++) /* scan along x */
{ if (swP == 0)
    { xp = xi - fpt->xs;
      yp = fpt->ys
        - (frho - xp*cos(ftheta))/sin(ftheta);
      if (yp < 0)
          yi = (int)(yp - 0.5);
      else
          yi = (int)(yp + 0.5);
    }
    else
    { xp = xi - sb.xs;
      yp = sb.ys - (frho - xp*cos(ftheta))/sin(ftheta);
      if (yp < 0)
          yi = (int)(yp - 0.5);
      else
          yi = (int)(yp + 0.5);
    }
    if ((yi >= ymin) && (yi < ymax))
        write_pixel(x+xi, y+yi, MAXPIX);
    } /* endfor xi */
else /* |slope| > 1 */
for (yi=ymin; yi<ymax; yi++) /* scan along y */
{ if (swP == 0)
    { yp = fpt->ys - yi;
      xp = fpt->xs
        + (frho - yp*sin(ftheta))/cos(ftheta);
      if (xp < 0)
          xi = (int)(xp - 0.5);
      else
          xi = (int)(xp + 0.5);
    }
    else
    { yp = sb.ys - yi;
      xp = sb.xs + (frho - yp*sin(ftheta))/cos(ftheta);
      if (xp < 0)
          xi = (int)(xp - 0.5);
      else

```

```

        xi = (int)(xp + 0.5);
    }
    if ((xi >= xmin) && (xi < xmax))
        write_pixel(x+xi, y+yi, MAXPIX);
    } /* endfor yi */
/* endif slope */
} /* endelse slanting */
} /* endfor fpt */
/* endfor levrow */
} /* end ReMapLevel */

/*****
| write the pyramid lines(in terms of actual rho,theta values) in a file
| INPUT : a level
*****/
void WritePyrPspace(levelmark *pyr)
{ FILE *ofp; /* output file */
  levelmark *lhead; /* head of level components of images */
  rowmark *rhead; /* head of rows of lines */
  lineseg *shead; /* a line segment */
  int i, j;

  DeriveName("pyr");
  /* open output file for writing */
  if ((ofp = fopen(name, "w")) == NULL)
  { printf("fopen failed on %s0, name);
    exit(1);
  }
  else
    printf("param space lines listed in %s0, name);

  /* write the pyramid summary */
  fprintf (ofp, "final pyramid: %1d levels, size = %1d
            pyr->levelnum, pyr->levelsub0);
  fprintf (ofp, "summary: maxlen, numH, numV, numN, numT0);
  for (j=0; j<(pyr-levelnum + 1); j++)
  { if (j==0)
    fprintf (ofp, "pyramid: ");
    else
    fprintf (ofp, "level %1d: ", j);
    for (i=0; i<5; i++)
    fprintf (ofp, " %3d ". summary[j][i]);
    fprintf (ofp, "0);
  }
  fprintf (ofp, "0);

  /* write each level's line segments */
  fprintf(ofp, "line: midpt(ys,xs) parameters(rho,theta) ");
  fprintf(ofp, "length(len) range(yp,xp)0);
  for (lhead=pyr; lhead !=NULL; lhead = lhead->nextlevel)
  { fprintf(ofp, "***Level %1d0, lhead->levelnum);

```

```

levrdel = lhead->levelrmodel;
levtdel = lhead->levelthetadel;
/* for each row of line segments */
for (rhead=lhead->level; rhead != NULL; rhead = rhead->nextrow)
  /* for each line segment */
  for (shead= rhead->row; shead != NULL; shead = shead->next)
    if (shead->len > 0)
      { fprintf(ofp, "m(%6.2f, %6.2f) ", shead->ys, shead->xs);
        fprintf(ofp, "p(%8.3lf, %8.5lf) l(%3d) ",
          shead->rhos *levrdel, shead->thetas *levtdel, shead->len);
        fprintf(ofp, "r(%3.2f, %3.2f)0,
          shead->subseg->yp, shead->subseg->xp);
      }
    /* endfor shead */
  /* endfor rhead */
} /* endfor lhead */
fprintf(ofp, "END0);
fclose(ofp);
} /* end WritePyrPspace */

/*****
* routines for printing out the lines
*****/

void PrintOneSeg(lineseg *seg);
void PrintRowSegs(rowmark *levrow, int j, int i);

/*****
| print one line segment
| INPUT : the line segment
*****/
void PrintOneSeg(lineseg *seg)
{ featpt *member; /* a subsegment */
  int n;

  printf("***(%6.2f, %6.2f) (%10.5lf, %10.5lf) (%1d)",
    seg->ys, seg->xs, seg->rhos * levrdel, seg->thetas * levtdel,
    seg->len);
  for (member=seg->subseg,n=0; member != NULL; member=member->next, n++)
    { if (n%5 == 0)
      printf(" ");
      printf("(%6.2f,%6.2f)", member->yp, member->xp);
    }
  printf("0);
} /* end PrintOneSeg */

```

```

/*****
| Print all segments found in row
| INPUT : a row of segments
*****/
void PrintRowSegs(rowmark *levrow)
{ lineseg *shead; /* a line segment */

  printf("rowsegments : ys=%.2f :0, levrow->ys);
  for (shead=levrow->row; shead != NULL; shead = shead->next)
    PrintOneSeg(shead);
} /* end PrintRowSegs */

/*****
| Print all segments found in level
| INPUT : a level
*****/
void PrintLevel(rowmark *level)
{ rowmark *rhead; /* head of row of line segments */
  lineseg *shead; /* a line segment */

  printf("level's segments :0);
  for (rhead=level; rhead != NULL; rhead = rhead->nextrow)
    PrintRowSegs(rhead, j, i);
} /* end PrintLevel */

/*****
| print the pyramid
| INPUT : the complete hierarchy
*****/
void PrintPyramid(levelmark *pyr)
{ levelmark *lhead; /* points to current level */

  for (lhead=pyr; lhead !=NULL; lhead = lhead->nextlevel)
  { levrdel = lhead->levelrhodel;
    levtdel = lhead->levelthetadel;
    printf("level %ld : " , lhead->levelnum);
    PrintLevel(lhead->level, 0, 0);
  }
} /* end PrintPyramid */

/***** end hhout.c *****/

```


APPENDIX I

THE PROGRAM SOURCE CODE FOR THE POSTPROCESSING
ROUTINES FOR THE HIERARCHY OF LINES
GENERATED BY HHOUGH

```

/*****/
* hhpost.c
* postprocessing of the lineseg pyramid formed by the hierarchical
* approach to the HT method, involves:
*   determination of midpoint and length
*   rho, theta parameters adjusted to be referenced on the image center
*   subseg field of lineseg node to contain:
*     expanse (ywidth, xwidth) of slanting lines (non-horizontal,
*       non-vertical); (imagesize,0) for verticals and (0,imagesize)
*       for horizontals;
*   all member subsegments are then deleted
* INPUT : size of image(dx,dy); and the hierarchy of lines, *pyramid.
*****/
#include "simpp2.h"
#include "s2hhdefs.h"
#include <math.h>
#include <alloc.h>

/* internal routines */
void GetEnds(lineseg *fpt, int hsize, int dx, int dy);
void CheckLimits(int dx, int dy);
void WritePyrAspace(levelmark *pyr, int summary[][5]);
void DeriveName(char *ext);

/* globals declared in hhough.c */
extern int sub0;          /* subimage size */
extern bounds sb;       /* subimage bounds markers */
extern double rhodel, thetadel; /* rho-theta sampling intervals */
extern double levrdel,levtdel; /*lev's rhodel,thetadel for accuvals->parvals*/

/* input image file name to be used for deriving outfile name */
/* declared in driver program for the image processing system or */
/* the driver program for testing the hhough */
extern char name[];

/* vars common to hhpost, ReMapLevel */
double halftdel; /* half the thetadel of level */
int halfsize; /* half the initial subimage size */
double frho, ftheta; /* rho, theta values in parameter space */
int xmin, xmax, ymin, ymax; /* bounds for area containing the lineseg */

/*****/
*****/
void hhpost(int dx, int dy, levelmark *pyramid)
{ featpt *tempsub, *marksub; /* temporary vars used for deleting subseg list*/
  levelmark *pyrlev; /* a level */
  rowmark *levrow; /* a row of lines */
  lineseg *fpt; /* a line segment */
  int xi, yi; /* point coordinates in image space */
  float xp, yp; /* coordinates of point wrt subimage center */
  float xs, ys; /* to hold old subimage center reference of line */

```

```

float xdel, ydel; /* temporary variables */
int n, done;     /* loop control variables */

/* retrieve level's accumulator array parameters */
halftdel = pyramid->levelthetadel /2;
halfsize = pyramid->levelsub0 /2;
for (n = (pyramid->levelnum -1); n > 0; n--)
    halfsize = halfsize/2;

/* set up parameter intervals for whole image space */
if (dx > dy)
    sub0 = dx;
else
    sub0 = dy;
thetadel = (double)PI / (4*sub0);
rhodel = (double)((int)((sub0 * sin(thetadel/2)) * 10 + 0.5))/10;

/* get image center */
sb.xs = (float)(dx -1)/2;
sb.ys = (float)(dy -1)/2;

/* adjust each line segment's rho-theta parameters to image center */
for (pyrlev = pyramid; pyrlev != NULL; pyrlev = pyrlev->nextlevel)
{
    levrdel = pyrlev->levelrhodel;
    levtdel = pyrlev->levelthetadel;

    /* update levelmark data */
    pyrlev->levelsub0 = sub0;
    pyrlev->levelrhodel = rhodel;
    pyrlev->levelthetadel = thetadel;

    for (levrow = pyrlev->level; levrow != NULL; levrow = levrow->nextrow)
        for (fpt = levrow->row; fpt != NULL; fpt = fpt->next)
            /* store original subimage center reference */
            {
                xs = fpt->xs;
                ys = fpt->ys;
                /* get rho,theta values in parameter space terms */
                frho = fpt->rhos * levrdel;
                ftheta = fpt->thetas * levtdel;
                /* get line segments endpoints for midpt, length det'n */
                GetEnds(fpt, halfsize, dx, dy);

                /* ensure that there is at least one subseg node to contain*/
                /* xwidth, ywidth of area contg line when necessary */
                if (fpt->subseg == NULL)
                {
                    tempsub = (featpt *)malloc(sizeof(featpt));
                    if (tempsub == NULL)
                    {
                        printf("malloc failed in hhpost0);
                        exit (1);
                    }
                    tempsub->next = NULL;
                }
            }
        }
    }

```

```

    fpt->subseg = tempseg;
}

/* set first subseg node as zero for vertical/horizontal lines */
/* or the xwidth and ywidth of line region for slanting lines */
fpt->subseg->xp = fpt->subseg->yp = 0;

/* note lineseg midpt in (ys, xs) and length in (len) */
if (ftheta < halftdel)    /*** VERTICAL LINE ***/
{ xp = fpt->xs + frho;
  if (xp < 0)
    xi = (int)(xp - 0.5);
  else
    xi = (int)(xp + 0.5);
  fpt->xs = xi;           /*** line's midpoint ***/
  fpt->ys = ((float)ymin + ymax-1)/2;
  fpt->len = ymax - ymin; /*** line's length ***/
  fpt->thetas = 0;       /*** make theta exactly 0 ***/
  fpt->subseg->yp = (float)sub0; /* for uniformity only */
}
else if ((ftheta > (PI/2 - halftdel))
  && (ftheta < (PI/2 + halftdel))) /*** HORIZONTAL LINE ***/
{ yp = fpt->ys - frho;
  if (yp < 0)
    yi = (int)(yp - 0.5);
  else
    yi = (int)(yp + 0.5);
  fpt->ys = yi;           /*** line's midpoint ***/
  fpt->xs = ((float)xmin + xmax-1)/2;
  fpt->len = xmax - xmin; /*** line's length ***/
  fpt->thetas = (int)((PI/2) /thetadel +.5);
  fpt->subseg->xp = (float)sub0;
}
else /*** SLANTING LINE ***/
{ if ((ftheta > (PI/4 - halftdel))
  && (ftheta < (3*PI/4 + halftdel))) /* |slope| <= 1 */
  { /* find endpoints by scanning the x */
    xdel = ydel = xmax;
    xi = xmin;
    done = 0;
    n = 0;
    while (n<2)
    { while (done == 0)
      { xp = xi - fpt->xs;
        yp = fpt->ys - (frho - xp*cos(ftheta))/sin(ftheta);
        if (yp < 0)
          yi = (int)(yp - 0.5);
        else
          yi = (int)(yp + 0.5);
        if ((yi >= ymin) && (yi < ymax))
          break; /* one end point found */
      }
    }
  }
}

```

```

    if (n == 0)
        xi++;
    else
        xi--;
    if ((xi < xmin) || (xi >= xmax))
        done = 1;    /* no points found */
}
if ((n == 0) && (done == 0))
    /* one end of line found */
{
    xdel = xi;
    ydel = yi;
    /* start next check with other end */
    xi = xmax - 1;
    n++;
}
else
    n = 2;
}
if (xdel < (float)xmax)
    /* the endpoints of the line were found */
{
    xp = xi - xdel;
    yp = yi - ydel;
    if ((int)yp == 0)
        /* the line is horizontal */
        {
            fpt->thetas = (int)((PI/2) / thetadel + .5);
            fpt->ys = (float)yi;
            fpt->xs = (xi + xdel)/2;
            fpt->len = xi - (int)xdel + 1;
            fpt->subseg->xp = (float)sub0;
        }
    else
        /*** the line is slanting indeed ***/
        {
            /* the line's midpoint */
            fpt->xs = (xi + xdel)/2;
            fpt->ys = (yi + ydel)/2;
            /* xwidth of area containing the line */
            fpt->subseg->xp = (float)xi-xdel;
            /* ywidth of area containing the line */
            fpt->subseg->yp = (float)yi-ydel;
            /* determine the line's length */
            xdel = (xi - xdel) * (xi - xdel);
            ydel = (yi - ydel) * (yi - ydel);
            fpt->len = (int)(sqrt(((double)xdel)+((double)ydel))+.5);
            /* adjust the theta for the prevalent thetadel */
            fpt->thetas = (int)(ftheta/thetadel + .5);
        }
}
else
{
    fpt->len = 0; /* line has no points in its subimage */
    fpt->thetas = (int)(ftheta/thetadel + .5);
}
}

```

```

} /* endif |slope| <= 1 */
else /* |slope| > 1 */
/* find endpoint by making a y scan */
{ xdel = ydel = ymax;
  yi = ymin;
  done = 0;
  n = 0;
  while (n<2)
  { while (done == 0)
    { yp = fpt->ys - yi;
      xp = (frho - yp*sin(ftheta))/cos(ftheta)
        + fpt->xs;
      if (xp < 0)
        xi = (int)(xp - 0.5);
      else
        xi = (int)(xp + 0.5);
      if ((xi >= xmin) && (xi < xmax))
        break; /* one endpoint of line found */
      if (n == 0)
        yi++;
      else
        yi--;
      if ((yi < ymin) || (yi >= ymax))
        done = 1; /* no points of the line was found */
    }
    if ((n==0) && (done==0))
      /* one end of the line was found */
      { ydel = yi;
        xdel = xi;
        /* start next check with other end */
        yi=ymax-1;
        n++;
      }
    else
      n = 2;
  } /* end find endpoints */
if (ydel < (float)ymax)
/* the endpoints of the line were found */
{ xp = xi - xdel;
  yp = yi - ydel;
  if ((int)xp == 0)
    /* the line is vertical */
    { fpt->xs = xi; /* the line's midpoint */
      fpt->ys = (yi + ydel)/2;
      fpt->len = yi - (int)ydel + 1; /* the line's length */
      fpt->thetas = 0; /* make theta exactly 0 */
      fpt->subseg->yp = (float)sub0;
    }
  else /**** the line is slanting indeed (not vertical)****/
    { fpt->xs = (xi + xdel)/2; /* the line's midpoint */
      fpt->ys = (yi + ydel)/2;
    }
}

```

```

    /* xwidth of the area containing the line */
    fpt->subseg->xp = (float)xi-xdel;
    /* ywidth of the area containing the line */
    fpt->subseg->yp = (float)yi-ydel;
    /* determine the line's length */
    xdel = (xi - xdel) * (xi - xdel);
    ydel = (yi - ydel) * (yi - ydel);
    fpt->len = (int)(sqrt(((double)xdel)+((double)ydel))+.5);
    /* adjust theta to current thetadel */
    fpt->thetas = (int)(ftheta/thetadel + .5);
  }
}
else
{ fpt->len = 0; /* there were no points found for the line */
  fpt->thetas = (int)(ftheta/thetadel + .5);
}
} /* endelse slanting line */

/* release all other subsegs except the very first one which */
/* contains the area defining the expanse of line in image */
if ((fpt->subseg != NULL) && (fpt->subseg->next != NULL))
{ tempsub = fpt->subseg->next;
  fpt->subseg->next = NULL;
  while (tempsub != NULL)
  { marksub = tempsub->next;
    tempsub->next = NULL;
    free (tempsub);
    tempsub = marksub;
  }
}

/* adjust line segment's rho-theta parameters to image center */
xdel = sb.xs - xs;
ydel = ys - sb.ys;
if (((int)xdel != 0) || ((int)ydel != 0))
{ frho = frho - xdel*cos(ftheta) - ydel*sin(ftheta);
  if (frho < 0)
    fpt->rhos = (int)(frho/rhodel - .5);
  else
    fpt->rhos = (int)(frho/rhodel + .5);
}
} /* endfor fpt */
/* endfor levrow */
} /* endfor pyrlev */
} /* end hhpost */

```

```

/*****

```

```

/*****
| Get the upper left corner and lower right corner of the level 0 subimages
| of the member linesegments, to define the area formed by the lowest level
| subimages that contains the line.
| CALLED BY : hhpost, ReMapLevel
| INPUT   : the line segment, *fpt; half initial subimage size, hsize;
|           xwidth, ywidth of area containing the line, (dx,dy)
*****/
void GetEnds(lineseg *fpt, int hsize, int dx, int dy)
{ featpt *head, *test; /* pointers to subsegment list */
  featpt min, max; /* upperleft corner/lowerright corner of area contg line*/

  if (fpt->subseg == NULL)
    /* the line has no subsegments */
    { min.xp = max.xp = fpt->xs;
      min.yp = max.yp = fpt->ys;
    }
  else
    /* scan the subsegment list */
    { head = fpt->subseg;
      min.xp = max.xp = head->xp;
      min.yp = max.yp = head->yp;
      test = head->next;
      /* find the upper left corner subimage */
      /* and the lower right corner subimage */
      while (test != NULL)
        { if (test->yp < (min.yp - .5))
            min.yp = test->yp;
          else if (test->yp > (max.yp + .5))
            max.yp = test->yp;
          if (test->xp < (min.xp - .5))
            min.xp = test->xp;
          else if (test->xp > (max.xp + .5))
            max.xp = test->xp;
          test = test->next;
        }
    }

  /* adjust endpts to encompass the level 0 subimage range involved */
  xmin = (int)(min.xp - hsize + 1);
  ymin = (int)(min.yp - hsize + 1);
  xmax = (int)(max.xp + hsize + 1);
  ymax = (int)(max.yp + hsize + 1);

  /* ensure that the endpoints are within the image space */
  CheckLimits(dx, dy);
} /* end GetEnds */

```



```

/*****
| Ensure that the bounds of the area containing the line are within the bounds
| of the image coordinates.
| CALLED BY : GetEnds, ReMapLevel
| INPUT   : size of the complete image (dx,dy)
*****/
void CheckLimits(int dx, int dy)
{
    if (xmin < 0)
        xmin = 0;
    if (ymin < 0)
        ymin = 0;
    if (xmax > dx)
        xmax = dx;
    if (ymax > dy)
        ymax = dy;
} /* end CheckLimits */

/*****
| Summary of the pyramid lines for each level and the overall pyramid:
| longest line; number of horizontals; number of verticals;
| number of slanting lines; total number of lines.
| CALLED BY : hthough
| INPUT   : a level
*****/
void SummarizeLevs(levelmark *pyr)
{ int summary[8][5]; /* 7levs: maxlen, numH, numV, numN, numT */
  levelmark *lhead; /* a level */
  rowmark *rhead; /* a rowlist */
  lineseg *shead; /* a line segment */
  int longest; /* longest line found */
  int i, j;

  /* initialize summary[][] */
  for (j=0; j<8; j++)
    for (i=0; i<5; i++)
      summary[j][i] = 0;

  for (lhead=pyr; lhead !=NULL; lhead = lhead->nextlevel)
  { j = lhead->levelnum; /* note the level number */
    longest = 0; /* length of longest line in level */
    for (rhead=lhead->level; rhead != NULL; rhead = rhead->nextrow)
      for (shead= rhead->row; shead != NULL; shead = shead->next)
        if (shead->len > 0)
          { if (shead->subseg->xp >= (float)sub0)
              summary[j][1] += 1; /* horizontal */
            else if (shead->subseg->yp >= (float)sub0)
              summary[j][2] += 1; /* vertical */
            else summary[j][3] += 1; /* neither */
            if (shead->len > longest)
              longest = shead->len;
          }
  }
}

```

```

    }
    /* endfor shead */
/* endfor rhead */

/* find the longest line at each level */
summary[j][0] = longest;
for (i=1; i<4; i++)
    summary[j][4] += summary[j][i];
} /* endfor lhead */

/* overall pyramid : summary[0][] */
longest = 0; /* max length in pyramid */
for (j=1; j<(pyr->levelnum + 1); j++)
{ for (i=1; i<5; i++)
    summary[0][i] += summary[j][i];
  if (summary[j][0] > longest)
    longest = summary[j][0];
}
summary[0][0] = longest;

/* write lines in the order of occurrence in the pyramid levels */
/* line attributes: orientation(Horizontal, Vertical, Neither); */
/* midpt(y,x); par(rhos,thetas) in terms of accumulator */
/* space quantization intervals; length */
WritePyrAspace(pyr, summary);

printf("Summary: level mlen numH numV numN numT ");
for (j=0; j<(pyr->levelnum + 1); j++)
{ printf("%4d ", j);
  for (i=0; i<5; i++)
    printf("%3d ", summary[j][i]);
  printf(" ");
}
} /* end SummarizeLevs */

/*****
| write the pyramid lines(in terms of rho,theta accu space values) in a file
| with the appropriate summary for each level: maxlen, number of horizontals,
| number of verticals, number of slanting lines, total number of lines.
| CALLED BY : hhough
| INPUT : the postprocessed pyramid, *pyr; the summaries
*****/
void WritePyrAspace(levelmark *pyr, int summary[][5])
{ FILE *ofp; /* output file */
  levelmark *lhead; /* a level */
  rowmark *rhead; /* a row of line segments */
  lineseg *shead; /* a line segment */
  int i;

/* derive name of output file from the input image name.
DeriveName("pas");

```

```

/* open output file for writing */
if ((ofp = fopen(name, "w")) == NULL)
{ printf("fopen failed on %s0, name);
  exit(1);
}
else
  printf("accu space lines listed in %s0, name);

/* write the format notes on the first 4 lines */
fprintf(ofp, "F P_yramid numlevs size maxlength numH numV numN numT0);
fprintf(ofp, "F L_level levnumber maxlength numH numV numN numT0);
fprintf(ofp, "F HVN ys xs rhos thetas length ywidth xwidth0);
fprintf(ofp, "F C_onversionfactors rhodel thetadel (for rhos, thetas)0);

/* write: C rhodel thetadel : rho=rhos*rhodel, theta=thetas*thetadel */
fprintf(ofp, "C %lf %lf0, pyr->levelrhodel, pyr->levelthetadel);

/* write pyr summary: P numlevs imagesize maxlen numH numV numN numT*/
fprintf(ofp, "P %1d %3d ", pyr->levelnum, pyr->levelsub0);
for (i=0; i<5; i++)
  fprintf(ofp, "%3d ", summary[0][i]);
fprintf(ofp, "0);

/* write level summary: L levelnum maxlen numH numV numN numT*/
for (lhead=pyr; lhead !=NULL; lhead = lhead->nextlevel)
{ fprintf(ofp, "L %1d ", lhead->levelnum);
  for (i=0; i<5; i++)
    fprintf(ofp, "%3d ", summary[lhead->levelnum][i]);
  fprintf(ofp, "0);

/* write out lines as: HVN ys xs rhos thetas len yp xp */
for (rhead=lhead->level; rhead != NULL; rhead = rhead->nextrow)
  for (shead= rhead->row; shead != NULL; shead = shead->next)
    if (shead->len > 0)
      { if (shead->subseg->xp >= (float)sub0)
          fprintf(ofp, "H "); /* horizontal */
        else if (shead->subseg->yp >= (float)sub0)
          fprintf(ofp, "V "); /* vertical */
        else
          fprintf(ofp, "N "); /* neither */
        fprintf(ofp, "%6.2f %6.2f %4d %3d %3d ", shead->ys,
          shead->xs, shead->rhos, shead->thetas, shead->len);
        fprintf(ofp, "%6.2f %6.2f0,
          shead->subseg->yp,shead->subseg->xp);
      }
    /* endfor shead */
  /* endfor rhead */
} /* endfor lhead */
fprintf(ofp, "F end0);
fclose(ofp);
} /* end WritePyrAspace */

```

```

/*****
| Final cleaning up.
| Release all nodes of the pyramid: subsegs, linesegs, rowmarks, levelmarks
| CALLED BY : hhough
| INPUT   : the pyramid, *pyr
|*****/
levelmark *FreePyramid(levelmark *pyr)
{ levelmark *lhead; /* a level */
  rowmark *rhead, *rmark; /* a row of line segments */
  lineseg *shead, *smark; /* a line segment */
  featpt *fhead; /* a subsegment */

  for (lhead=pyr; lhead !=NULL; lhead = lhead->nextlevel)
  { for (rhead=lhead->level; rhead != NULL; rhead = rhead->nextrow)
    { shead= rhead->row;
      rhead->row = NULL; /* detach linesegs from rowmark */
      while (shead != NULL)
      { smark = shead->next;
        shead->next = NULL; /* detach head of lineseg list */
        fhead = shead->subseg;
        shead->subseg = NULL; /* detach subseg (only one left) */
        free(fhead); /* free subseg */
        free(shead); /* free lineseg */
        shead = smark; /* next lineseg in list */
      }
    } /* endfor rhead */
    /* delete rowmarks */
    rhead = lhead->level;
    lhead->level = NULL; /* detach rowmarks from level */
    while (rhead != NULL)
    { rmark = rhead->nextrow;
      rhead->nextrow = NULL; /* detach head of rowmarks list */
      free(rhead); /* free the rowmark */
      rhead = rmark; /* next rowmark in list */
    }
  } /* endfor lhead */
  /* delete levelmarks */
  while (pyr != NULL)
  { lhead = pyr->nextlevel;
    pyr->nextlevel = NULL; /* detach head of levelmarks */
    free(pyr); /* free topmost levelmark */
    pyr = lhead; /* next levelmark */
  }
  return (pyr);
} /* end FreePyramid */

/*****
| derive name of output file from name of input file by changing the .ext
|*****/
void DeriveName(char *ext)
{ int i, j;

```

```
/* derive name for output file : name.pyr */  
i = j = 0;  
while (name[i] != '.')  
    i++;  
i++;  
while ((name[i] = ext[j]) != ' ' )  
{    i++;  
    j++;  
}  
} /* end DeriveName */
```

```
***** end hhpost.c *****/
```

APPENDIX J

A SAMPLE OUTPUT OF THE POSTPROCESSED HIERARCHY OF LINES

F P_ramid numlevs size maxlength numH numV numN numT
 F L_level levnumber maxlength numH numV numN numT
 F HVN ys xs rhos thetas length ywidth xwidth
 F C_onversionfactors rhodel thetadel (for rhos, thetas)
 C 0.400000 0.003068
 P 5 256 256 13 20 9 42
 L 5 256 2 10 3 15
 N 26.50 133.50 233 503 211 5.00 211.00
 V 111.50 4.00 -309 0 144 256.00 0.00
 V 119.50 80.00 -119 0 32 256.00 0.00
 V 111.50 103.00 -61 0 48 256.00 0.00
 V 119.50 100.00 -69 0 64 256.00 0.00
 V 119.50 255.00 319 0 240 256.00 0.00
 V 103.50 254.00 316 0 192 256.00 0.00
 V 119.50 146.00 46 0 32 256.00 0.00
 V 111.50 217.00 224 0 80 256.00 0.00
 V 111.50 192.00 161 0 48 256.00 0.00
 V 119.50 209.00 204 0 64 256.00 0.00
 N 205.00 79.50 -211 522 127 -4.00 127.00
 N 210.50 135.50 -228 525 239 -9.00 239.00
 H 238.00 127.50 -296 512 256 0.00 256.00
 H 239.00 127.50 -299 512 256 0.00 256.00
 L 4 127 1 1 5 7
 N 1.00 47.50 290 502 95 2.00 95.00
 N 55.50 35.50 -226 8 63 63.00 1.00
 H 94.00 47.50 63 512 96 0.00 256.00
 N 1.50 191.50 293 516 127 -1.00 127.00
 N 8.00 191.50 285 500 95 4.00 95.00
 V 55.50 148.00 51 0 64 256.00 0.00
 N 181.00 39.50 -149 520 79 -2.00 79.00
 L 3 48 2 0 0 2
 H 66.00 223.50 137 512 32 0.00 256.00
 H 194.00 39.50 -186 512 48 0.00 256.00
 L 2 32 4 5 0 9
 H 3.00 111.50 290 512 32 0.00 256.00
 V 39.50 166.00 96 0 32 256.00 0.00
 V 135.50 28.00 -249 0 32 256.00 0.00
 V 135.50 32.00 -239 0 32 256.00 0.00
 V 135.50 77.00 -126 0 32 256.00 0.00
 H 157.00 47.50 -99 512 32 0.00 256.00
 H 215.00 23.50 -233 512 16 0.00 256.00
 H 199.00 239.50 -192 512 32 0.00 256.00
 V 227.50 4.00 -309 0 24 256.00 0.00
 L 1 16 4 4 1 9
 H 20.00 23.50 249 512 16 0.00 256.00
 V 47.50 167.00 99 0 16 256.00 0.00
 H 70.00 231.50 124 512 16 0.00 256.00
 H 94.00 135.50 64 512 16 0.00 256.00
 N 163.50 103.50 -107 528 15 -1.00 15.00
 V 159.50 148.00 51 0 16 256.00 0.00
 V 159.50 216.00 221 0 16 256.00 0.00

V 175.50 166.00 96 0 16 256.00 0.00
H 201.00 183.50 -204 512 16 0.00 256.00
F end