VISION SYSTEMS FOR A MOBILE ROBOT BASED ON LINE DETECTION USING THE HOUGH TRANSFORM AND ARTIFICIAL NEURAL NETWORKS

GIDEON KANJI DAMARYAM

A thesis submitted in partial fulfilment of the requirements of
The Robert Gordon University
for the degree of Doctor of Philosophy

November 2008

Declaration

I confirm that the material in this report is my own work. Where this is not the case, the source of material has been acknowledged.

Signed ……………………………

Abstract

This project contributes to the problem of mobile robot self-navigation within a rectilinear framework based on visual data. It proposes a number of vision systems based on detection of straight lines in images captured by a robot using the Hough transform and artificial neural networks as core algorithms. The Hough transform is a robust method for detection of basic features (Boyce et al 1987). However, it is so computationally demanding that it is not commonly used in real time applications and applications which utilise anything but small images (Song and Lyu 2005). (Dempsey and McVey 1992) have suggested that this problem might be resolved if the Hough transform were implemented with artificial neural networks. This project investigates the feasibility of systems using these core algorithms, and systems that are hybrids of them.

Prior to application of the core algorithms to a captured image, various stages of pre-processing are carried out including resizing for optimum results, edge-detection, and edge thinning using an adaptation of the thinning method of (Park, 2000) proposed by this work. An analysis of the costs and benefits of thinning as part of pre-processing has also been performed.

The Hough transform based system, which has been largely successful, has involved a number of new approaches. These include a peak detection scheme; post-processing schemes which find valid sub-lines of lines found by the peak detection process, and establish which high-level features these sub-lines represent; and an appropriate navigation scheme.

Two artificial neural network systems were designed based on lines detection and sub-lines detection respectively. The first was able to detect long lines, but not shorter (even though navigationally important) lines, and so was aborted. The second system has two major stages. Networks of stage 1 developed to detect sub-lines in sub-images derived by breaking down the original images, did so passibly well. A network in stage 2 designed to use the results of stage 1 to guide the robot's motion did not do so well for most test images. The networks of stage 1, however, have been helpful with development of a hybrid vision system.

Suggestions have been made on how this work can be furthered.

Acknowledgements

I am profoundly grateful to my Director of Studies, Graeme Dunbar for his support and guidance through the duration of the course. I am also grateful to my other supervisors Dr Chris Aust and Grant Maxwell. Special thanks to Dr Chris MacLeod for the major role he played towards the winding up stage of the project, and to other members (and past members) of the School of Engineering notably, Dr Ann Reddipogu, Dr Stephenie Rigby, and my former fellow PhD student Dr Ananda Jagadeesan. A big thank you also goes to Martin Simpson and Rosy Mearns of the Research Support Office, and Euan Fergusson formerly of the International Office.

I am deeply indebted to the Robert Gordon University for the studentship without which I could not have been on this course, and to my employers, the Nigeria National Petroleum Corporation for releasing me to attend the course and taking me back afterwards.

Lastly, I would like to thank family and friends too numerous to mention individually for all their love, encouragement and support, and distractions. In particular, I would like to thank Mallam Jibrin Wunti for helping with formatting some of the mathematical expressions, Patience Nuhu for proof reading some chapters, and Deborah Dadi whose interest and concern has been overwhelming.

Table of Contents

Declaration .... **i**

Abstract .... **ii**

Acknowledgements .... **iii**

Table of Contents .... **iv**

List of Figures .... **viii**

List of Tables .... **xi**

Chapter 1 Introduction **1**

1.1 Project Title and Key Concepts 1
1.2 Introduction to Complexity Analysis of Algorithms 2
1.3 Strengths and Weaknesses of the Hough Transform and Neural Networks 4
1.3.1 Strengths and Problems of the Hough Transform 4
1.3.2 Strengths and Problems of Artificial Neural Networks 4
1.4 A Hybrid Approach as a Possible Improvement on Both Approaches 5
1.5 Experimental Equipment and Tools 6
1.5.1 Robot 6
1.5.2 Other Computers Used 7
1.5.3 Software Development Tools 7
1.5 Test Environment 8
1.6.1 Contrast between Floor and Wall 8
1.6.2 Varying Light Conditions 8
1.6.3 Reflections 9
1.6.4 Doors 10
1.6.5 Other Objects 10
1.7 Project Objectives 11
1.8 New Ideas Developed in this Work 11
1.8.1 Modified Thinning Algorithm 12
1.8.2 Effects of Thinning 12
1.8.3 Automatic Determination of Threshold for Peak Detection 12
1.8.4 Peak Detection Scheme 12
1.8.5 Actual-Lines Extraction Scheme 13
1.8.6 Line Categorisation Scheme 13
1.8.7 Features Recognition Schemes 13
1.8.8 Navigation Algorithms 13
1.8.9 Neural Network Scheme for Detecting Lines 13
1.8.10 Hybrid Vision Systems 14
1.9 Outline of Other Chapters 14

Chapter 2 Background **16**

2.1 The Hough Transform 16
2.1.1 Input Images for the Hough Transform 16
2.1.2 Pre-Processing 17
2.1.3 Transformation from Image Space to Hough Space 19
2.1.4 The Accumulator Array 22
2.1.5 Accumulation 23
2.1.6 Peak detection 23
2.1.8 Reconstruction 26
2.2 Artificial Neural Networks 28
2.2.1 A Quick Look at Biological Neurons and Neural Networks 29
2.2.2 Artificial Neurons and Neural Networks 31
2.2.3 Types of Artificial Neural Networks 32
2.2.4 The Back Propagation Network 33
2.2.5 Applications of Back Propagation Networks 35

Chapter 3 Related Work 37

3.1 Works Related to the Performance of the Hough Transform Algorithm 37
3.1.1 The Work of Princen and Others 37
3.1.2 The Work of Boyce and Others 38
3.1.3 The Work of Grimson and Others 39
3.2 The Hough Transform in Vision Systems for Mobile Robot Navigation 40
3.2.1 The Work of Tyson 40
3.2.2 The Work of Djekoune and Achour 40
3.2.3 The Work of Vaughn and Arkin 41
3.3 Artificial Neural Networks in Vision Systems for Mobile Robot Navigation 43
3.3.1 The Work of Chang and Others 43
3.3.2 The Work of Inigo and Others 44
3.4 Works which Employ a Combination of the Hough Transform and Artificial Neural Networks 45
3.4.1 The Work of Dempsey and McVey 45
3.4.2 The Work of Yun and Others 45
3.4.4 The Work of Meng and Kak 47
3.5 Other Related Works 49
3.5.1 The Work of Li and Others 49

Chapter 4 Pre-processing 50

4.1 Image Capture, Resizing and Conversion to Gray Scale 50
4.1.1 Capture-Process-Navigate Cycle 50
4.1.2 Capture 50
4.1.3 Resizing 51
4.1.4 Intensity Determination 53
4.1.5 Image Point Indexing 53
4.2 Edge Detection 53
4.2.1 Edge Threshold determination 55
4.3 Edge Thinning 56
4.4 Effect of Thinning on Hough Transform Processing Time and Results 59
4.4.1 Effects of Thinning on Quality of Results of Subsequent Processes 59
4.4.2 Effects of Thinning on Processing Times of Subsequent Processes 62

Chapter 5 A Hough Transform Vision System for a Mobile Robot 66

5.1 Implementation of the Hough Transform 66
5.1.1 Coordinates in Image Space 67
5.1.2 θ Resolution 67
5.1.3 Range of the Hough Transform Function 69
5.1.4 Accumulator Array 70
5.1.5 Transformation and Accumulation 71
5.1.6 Peak Detection 71
5.2 Post Processing 83
5.2.1 Determination of Actual Lines 83
5.2.3 Vanishing Point Estimation 93
5.2.4 Category and Position Assignment to Lines 98
5.3 High Level Features Determination 104
5.3.1 Corridor(s) Recognition 104
5.3.2 Door(s) Recognition 106
5.4 Navigation 109
5.4.1 Move-Along-Corridor 109
5.4.2 Turn into Wall-Door 110

Chapter 6 Mobile Robot Vision Systems based on Line Detection using Artificial Neural Networks 111

6.1 Training Parameters for Back-Propagation Networks Used 112
6.1.1 Outer Loop 113

6.1.2 Inner Loop 113
6.2 Robot Vision System based on Full Line Detection 114
6.2.1 Vertical Category Lines Recognition with Full Line Detection 115
6.2.2 Vertical Backslash Category Lines Recognition 116
6.2.3 Conclusion 118
6.3 Vision for Navigation based on Sub-Images Processing 119
6.3.1 Recognition of Lines in Categories from Sub-Images 120
6.3.2 Processing 'Full Picture' 131

Chapter 7 Hybrid Hough Transform/Neural Networks Vision Systems for a Mobile Robot 132

7.1 Hybrids with Artificial Neural Networks in Hough Transform System 132
7.1.1 Hybrid 1: Use of an ANN to Map from Image Space to Peaks in Accumulator Array 132
7.1.2 Hybrid 2: Use of an ANN for Peak Detection 133
7.1.3 Hybrid 3: Use of an ANN to Map from Peaks in Binary Accumulator Array to Vanishing Point 133
7.2 Hybrids with the Hough Transform in Artificial Neural Networks System 133
7.2.1 Hybrid 4: Use the Hough transform to find line segments in sub-images and then use ANN to establish full picture 134
7.2.2 Hybrid 5: Use ANNs to find lines in Sub-Images and then use Hough transform to establish 'full-picture' 135
7.3 Summary 142

Chapter 8 Summary, Conclusions and Suggestions for Future Work 143

8.1 Introduction to Chapter 143
8.2 Background and Preliminary Achievements 143
8.2.1 Environment 143
8.2.2 Background 144
8.2.3 Related Work 144
8.2.4 Pre-Processing 144
8.3 Achievements and Conclusions Related to a Mobile Robot Vision System based on Line Detection using the Hough transform 145
8.3.1 Customisation and Implementation of the Hough Transform 145
8.3.2 Vanishing Point Estimation 146
8.3.3 Line Categorisation 146
8.3.4 High-Level Features Determination 146
8.3.5 Navigation 147
8.4 Achievements and Conclusions Related to a Mobile Robot Vision System based on Line Detection using Artificial Neural Networks 147
8.5 Achievements and Conclusions Related to a Hybrid Hough Transform/Artificial Neural Networks Mobile Robot Vision System 148
8.6 Summary of Suggestions for Future Work 148

References 150

Appendix A Publication Produced During Research 153

Appendix B Details of Results of Sub-Lines Determination for a Sample Image 154

Appendix C Artificial Neural Network Training Set and Sample Test Results for Vertical Category 156

Training Set for Vertical Line Recognition 156
Sample Test Results 167

Appendix D Sample Sub-Image Line Recognition Results for a Full Image 224

Sample Thinned Binary Image 224
Results from Sample Image 224

Testing Set......*225*
Results Vectors......*234*

Appendix E Sample Source Code......236

File “point.h” Contents......236
File “HT_point.h” Contents......236
File “image.h” contents......237
File “subResults.h” Contents......239
Some Main Source File Contents......241

Appendix F Technical Specifications of the Koala Robot......260

Appendix G Time Complexity Analysis for Some Processes in Hough Transform System......261

Conversion to gray scale......261
Edge Detection......261
Edge Thinning......261
Hough Transform......261
Peak Detection......261
Determination of Sub-Lines......262

List of Figures

Figure 1. 1 Koala robot used for current work 6
Figure 1. 2 Contrast between floor and wall........ 8
Figure 1. 3 Effects of lighting 9
Figure 1. 4 Alterations due to reflection 10
Figure 1. 5 Doors in the corridor environment 10
Figure 1. 6 A radiator in the corridor environment........ 11

Figure 2.1 A biological neuron........ 30
Figure 2.2 A synapse 31
Figure 2.3 Neurons arranged in layers in a typical back propagation network 34

Figure 3.1 The butterfly filter 38
Figure 3.2 The reduced butterfly filter 38
Figure 3.3 Example of a Corridor Node in (Meng and Kak 1993)........ 48

Figure 4.1 Effects of various image sizes........ 52
Figure 4.2 Image points indexing 53
Figure 4.3 Sample Sobel edge detection results........ 56
Figure 4.4 Comparison of the results of the thinning method of (Park, 2000) and the modified version of it used in this work 58
Figure 4.5 Navigation critical lines in a typical image........ 62

Figure 5.1 Angle due to 1 pixel difference at farthest distance from origin........ 68
Figure 5.2 Lines drawn across the range of θ at 1° interval 69
Figure 5.3 Lines drawn across the range of θ at 2° interval 69
Figure 5.4 Ranges of θ and ρ in Accumulator Array........ 70
Figure 5.5 Navigation critical lines in a typical image – image1 72
Figure 5.6 Lines and sub-lines found in image1 with T=100 73
Figure 5.7 Lines and sub-lines found in image1 with T=150 73
Figure 5.8 Lines and sub-lines found in image1 with T=200 73
Figure 5.9 Lines and sub-lines found in image1 with T=250 74
Figure 5.10 Lines and sub-lines found in image1 with T=300 74
Figure 5.11 Navigation critical lines in a typical image – image2 75
Figure 5.12 Lines and sub-lines found in image2 with T=100 76
Figure 5.13 Lines and sub-lines found in image2 with T=150 76
Figure 5.14 Lines and sub-lines found in image2 with T=200 76

Figure 5.15 Lines and sub-lines found in image2 with T=250 77
Figure 5.16 Lines and sub-lines found in image2 with T=300 77
Figure 5.17 Navigation critical lines in a typical image – image3 78
Figure 5.18 Lines and sub-lines found in image3 with T=100 78
Figure 5.19 Lines and sub-lines found in image3 with T=150 78
Figure 5.20 Lines and sub-lines found in image3 with T=200 79
Figure 5.21 Lines and sub-lines found in image3 with T=250 79
Figure 5.22 Lines and sub-lines found in image3 with T=300 79
Figure 5.23 Parameter for selection criteria for valid sub-lines 87
Figure 5.24 Result of sub-line detection algorithm 90
Figure 5.25 Sample Vanishing points found.. 96
Figure 5.26 Determining magnitude of distance of line from vanishing point 100
Figure 5.27 Sample result of vanishing point determining algorithm 102
Figure 5.28 An extract of accumulator array showing some points found relative to the curve of accumulator array... 103
Figure 5.29 Sample result of corridor detection scheme 106
Figure 5.30 Sample door sub-lines found 108
Figure 5.31 Sample wall-door-right detected 108

Figure 6.1 Width of parallelogram containing full range of vertical category lines at ρ = 0.. 115
Figure 6.2 Parallelogram containing full range of vertical backslash category lines 117
Figure 6.3 Some vertical backslash lines not found by full line detecting neural network for the vertical backslash category 118
Figure 6.4 Pre-processed image broken down into 8x8 sized sub-images 119
Figure 6.5 Possible appearances of extreme vertical category lines in sub-images 121
Figure 6.6 Broken down image used for testing 122
Figure 6.7 Results of test for vertical category 122
Figure 6.8 Possible appearances of extreme vertical backslash category lines in sub-images 123
Figure 6.9 Results of test for vertical backslash category 123
Figure 6.10 Possible appearances of extreme backslash category lines in sub-images 124

Figure 6.11 Results of test for backslash category... 124
Figure 6.12 Possible appearances of extreme horizontal backslash category lines in sub-images... 125
Figure 6.13 Results of test for horizontal backslash category 125
Figure 6.14 Possible appearances of extreme horizontal category lines in sub-images 126
Figure 6.15 Results of test for backslash category... 126
Figure 6.16 Results of test for horizontal slash category.................................. 127
Figure 6.17 Results of test for slash category 127
Figure 6.18 Results of test for vertical slash category...................................... 128
Figure 6.19 Sample combined result of sub-lines detection in sub-image using ANNs 129

Figure 7.1 Vertical lines found in sub-images using ANNs................................ 136
Figure 7.2 Vertical lines found in the 'full picture' 137
Figure 7.3 Vertical lines with valid sub-lines found in the 'full picture' with hybrid 5 138

List of Tables

Table 4.1 Results of processing to the point of sub-lines detection from thinned and “unthinned” edge image 61
Table 4.2 Comparison of process times for an image when it is thinned and when it is not 63

Table 5.1 Processes information for image1 with various targets for number of edges 75
Table 5.2 Processes information for image2 with various targets for number of edges 77
Table 5.3 Processes information for image3 with various targets for number of edges 80
Table 5.4 Definitions for preliminary classes of lines 95
Table 5.5 Breakdown of scores for top 10 intersection points 97
Table 5.6 Some results from VP determination scheme 98
Table 5.7 Lines Categorisation 99
Table 5.8 θ and ρ values for lines found in a typical image 103
Table 5.9 Criteria for selection of left corridor edge 105
Table 5.10 Criteria for selection of right corridor edge 106

Table 6.1 Sub-image labelling order 119
Table 6.2 Configuration of results vector 129
Table 6.3 Colours used to indicate different lines found in a sub-image 130

Table 7.1 Samples of lines found by applying Hough transform to sub-images for different categories 135
Table 7.2 Parameter values for vertical lines found in the ‘full picture’ 137
Table 7.3 Sub-Images which made up valid vertical sub-lines 138
Table 7.4 Examples of inputs to hybrid 5 Hough transform, peaks found and sub-lines found 140
Table 7.5 Details of peaks found by hybrid 5 Hough transform 140
Table 7.6 Details of sub-lines found by hybrid 5 Hough transform 141

Chapter 1 Introduction

This chapter introduces the project. It begins with a statement of the title of the project and very brief explanations of the key concepts used in the project, namely, Hough transforms (HTs) and Artificial Neural Networks (ANNs). It also introduces analysis of time taken by algorithms in computer science, as such analysis is used in various later parts of this work. The rest of this chapter looks at the strengths and weaknesses of these key concepts, and explains the motivation for a hybrid of the two. It then discusses the objectives of this project and summarises some new ideas developed by this work towards achievement of the objectives. Finally, it gives an outline of other chapters in this thesis.

1.1 Project Title and Key Concepts

The title of this project is “Vision Systems for a Mobile Robot based on Line Detection using the Hough transform and Artificial Neural Networks”. It is a contribution to the problem of mobile robot self-navigation based on visual data. Hough transforms and Artificial Neural Networks are both techniques used in machine vision.

Hough transforms are used for detection of features such as lines, curves and simple shapes within images. They work by transforming a target feature in a given image to a point in a new image while accumulating a measure of the likelihood that a point in the new image is due to a feature of the required type from the original image. When the transformation is complete, points in the new image can then be subjected to a predefined threshold so that points that are very likely to be due to the required kind of feature can be selected and the original features identified by reversing the transformation process. The Hough transform used depends on the feature to be detected. As this work is concerned with the detection of straight lines, it uses the transform called the straight line Hough transform (SLHT). In this work, the SLHT is simply referred to as the Hough transform, as is commonly done. A more detailed introduction to the Hough transform is given in *2.1 The Hough Transform* and implementation of the Hough transform in this project is discussed in *Chapter 5 A Hough Transform Vision System for a Mobile Robot*.

Artificial neural networks (ANNs) or simply, neural networks (NNs), are widely used in pattern recognition. Typical patterns are fingerprints, faces, statistical data, or mathematical functions. ANNs work by imitating in some way, the functioning of neural structures in nature such as brains and nerves. There are many different types of ANNs. Typically, they consist of several artificial neurons which are mathematical models of biological neurons. Artificial neurons are ‘connected’ by some algorithm which also governs the training of the network and processing of data in the network. This work uses the type of ANNs called back-propagation networks (BPNs). They are probably the most common types of artificial neural networks in use, and often the expression artificial neural networks in literature actually refers to back-propagation networks. This attitude is adopted in this work, so the expression artifical neural network will mean back-propagation artificial neural networks except when it is necessary to make a distinction. A more detailed introduction to artificial neural networks in general and back-propagation networks in particular, is given in *2.2 Artificial Neural Networks*. Their use in this project is discussed in *Chapter 6 Mobile Robot Vision Systems based on Line Detection using Artificial Neural Networks*.

1.2 Introduction to Complexity Analysis of Algorithms

In computer science, algorithms are analysed using what is refered to as the Big O notation. This section briefly introduces this.

There are two key concerns when analysing the performance of algorithms. These are the amount of space the algorithms requires, and the amount of time it takes to run.

Space complexity analysis studies the use of computer memory by a running algorithm. In recent times, it is not usually very critical for typical applications because modern computers usually have vast memory. It is not usually considered as intently in most modern analysis of the performance of an algorithm, as time complexity. This attitude is used in this work – space use is considered where appropriate, but generally not as closely as time complexity.

Time complexity analysis studies the use of computer processor time by algorithms. The amount of time that different computers, or even the same computer at different times, will take to run an algorithm (with a constant number of operations) varies widely. One computer can accomplish in 0.32 milliseconds what another computer will need 516 milliseconds to accomplish, and another computer still may require a full minute (60,000) milliseconds to accomplish the same task. An important component of time analysis therefore, is estimating the number of core operations required for very large inputs if the algorithm runs with its worst case. This can be in addition to, or instead of analysis of the actual time taken for the algorithm to run. The number of inputs is commonly denoted as n, and the number of core operations performed is normally estimated as a function, O, of n. Mathematically this is written as $\mathrm{O}(n)$.

If a function has number of core tasks varying in direct proportional to n for example, the function is said to be of order n and this is written as

$$\mathrm{O}(n) = n \qquad . \qquad . \qquad . \qquad \text{Equation 1.1}$$

A very common example of an algorithm with linear time complexity is linear search which sequencially searches through elements of a list to find an element or elements matching a given criteria.

If on the other hand, the number of core tasks to be carried out by an algorithm varies logarithmically, then the function is said to be of order $\log_n$. This is written as

$$\mathrm{O}(n) = \log_n \qquad . \qquad . \qquad . \qquad \text{Equation 1.2}$$

A common example of an algorithm with such time complexity is binary search which searches a sorted array by searching the middle element of an array, and then the middle element of either the left or right sub-array depending on how the current element does against the given criteria.

Other measures of time complexity exist including Ω which is a measure of the performance of the algorithm when it runs for the best case, and Θ used for for the average case. These are not as commonly used as O.

Various orders of time complexity for algorithms are discussed in *Appendix G Time Complexity Analysis for Some Processes in Hough Transform System*.

1.3 Strengths and Weaknesses of the Hough Transform and Neural Networks

1.3.1 Strengths and Problems of the Hough Transform

The Hough transform is “inherently stable and robust” (Boyce et al 1987). It copes very well with noisy images (Dempsey and McVey 1992). It can find features even if the patterns representing them are broken in the original image (Atiquzzaman 1992).

However, the standard Hough transform typically requires a lot of computing time and space to implement. For this reason it is not commonly used in real time applications (Dempsey and McVey 1992) and image-processing applications which utilise anything but small images (Song and Lyu 2005).

Also, the standard Hough transform returns spurious maxima and so can be unreliable (Boyce et al 1987). It accumulates points from lines or other features which lie along the same direction in the original image, but are not actually parts of the same feature.

1.3.2 Strengths and Problems of Artificial Neural Networks

Artificial neural networks are very good at dealing with problems in which a large amount of input/output data is available, but where how the output can be worked out from the input is not known or is very complicated. An understanding of the input, the output, or the manner in which the outputs are derived from the inputs is not necessary.

ANNs can learn to ignore input components that are irrelevant to the solution. They can also cope with scene variations such as variations in level of illumination, and are capable of recognising scenes from low resolution images saving processing time and lowering hardware cost.

They can recognise patterns they have not encountered in training (Inigo et al 1995).

Initial development and training of a back-propagation network to perform a particular task can be very time consuming. In some situations, other techniques would do much better than back-propagation networks.

If the input/output set is fairly limited and discrete, and can be set-up in a look-up table of a reasonable size, then it is easier to use a look up table, and results would be more accurate.

ANNs are not good at giving precise numeric answers, so they are not suitable for use where precise numeric answers are required. If it is possible to develop an algorithm to solve a problem then it is generally better to develop and implement that algorithm rather than use an ANN, except if implementation of the algorithm is infeasible.

1.4 A Hybrid Approach as a Possible Improvement on Both Approaches

In the context of hardware implementations, (Dempsey and McVey 1992) have suggested that the computational demand problem of the Hough transform might be resolved if implementations of the Hough transform with artificial neural networks were possible. They have gone on to developed an ANNs-like circuitry to map from image space to parameter space, and detect peaks in Hough transform parameter space (two important but time consuming steps in the Hough transform). In (Dempsey and McVey 1991) they discuss enhancements to the artificial neural network/Hough transform system such as multiple feature detection by a relatively simple hardware addition to the basic system. They report significant improvement in time usage.

This project sets out to investigate the feasibility of Hough transform/artificial neural network hybrids using software simulations. It begins by separately developing vision systems based on line detection using each of the Hough transform and artificial neural networks.

1.5 Experimental Equipment and Tools

The experiments in this work were carried out with equipment and tools including a Koala robot, a desktop personal computer, a laptop, and software tools including Borland builder versions 5 and 6, a software plugin called video lab, and a library of object oriented C++ neural network classes adapted from (Rogers 1996). The sub-sections which follow elaborate further.

1.5.1 Robot

The Koala robot is a mobile robot developed by the K-team, a Switzerland based team of roboticist, primarily for research. A picture of it is shown in figure 1.x

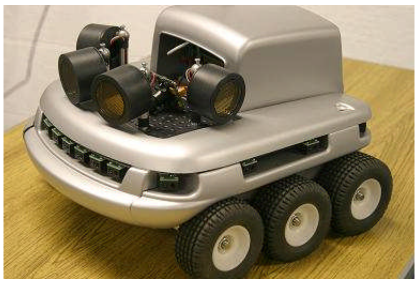


Figure 1. 1 Koala robot used for current work

It moves with the aid of 2 motors which drive 6 wheels – 3 on each side of the robot.

1.5.1.1 Hardware of the Koala

The koala has a motorola 68331 processor with 22MHz clock speed. More details of the technical specifications of the Koala robot are available in *Appendix F Technical Specifications of the Koala Robot*

The koala used for this work was fitted with various CCD cameras at different times. Additionally a sony DSC10 digital camera was also used to capture images for the purpose of the work.

The koala robot used for this project was also equipped with a PC-104 extension. The PC 104 extension has the following features:

- Pentium MMX 266 MHz with 64MB RAM
- Onboard Video Display
- Onboard Ethernet Interface
- EIDE, Floppy, keyboard, mouse and SVGA monitor interface
- PC/104 and PC/104+ extension bus
- Ultra Slim ATA 20Gb disk (2.5")

- Power Converter for Koala

Source: http://www.k-team.com/robots/koala/pc104.html

The PC 104 extension provides more computer power for the koala robot. It is loaded with Red Hat Linux and makes it possible to interact with the koala via Linux and the C++ development environment it provides rather than actually interacting with the BIOS of the Motorola 68331 system. It also makes it possible to attach peripherals such as monitors, mouse and keyboard, and to connect to another PC using a LAN interface.

The PC 104 is connected to the base Motorola 68331 system of the koala via a serial connection.

1.5.1.2 Software library which shipped with the Koala

The PC104 was shipped with a number of C++ classes available for programmers of the robot. Three were used in this project – the Robot class, the Camera class and the Wheels.

The robot class contains the other two classes, and also provides functions for communication with external systems. The camera class provides the capability to caption images. The wheels class provides public functions which make it possible to set the speed of the wheels and read the current speeds.

Several other classes shipped with the koala but will not be mentioned as they are not used in this project.

1.5.2 Other Computers Used

Three other PCs were used along with the Koala - a Pentium III desktop personal computer, a Pentium IV Advent 3.0 GHz laptop with model number 1046 and a Sony Viao Centrino Duo Core 1.83GHz laptop with model number VGN-SZ3HP/B.

1.5.3 Software Development Tools

The software development tools used in this work include Borland C++ Builder versions 5 and 6, a software plugin called Video Lab deveped at MIT and available free at www.mitov.com, and a library of object oriented C++ neural network classes adapted from (Rogers 1996)

Sample source code developed is included in *Appendix E Sample Source Code*.

1.5 Test Environment

In order to test ideas, a constrained but realistic scenario was developed in which a small mobile robot was used to navigate a real environment. A corridor was selected as it offered strong but natural features.

The sub-sections which follow highlight some visual clues and other issues that are important in such a scenario.

1.6.1 Contrast between Floor and Wall

In the corridors used for this work, the walls are white and have horizontal and vertical grooves in them, while the floor is reddish brown in colour as illustrated in Figure 1.2. Figure 1.2a is an image taken by the robot across the corridor, and 1.1b is taken at an acute angle to one of the walls of the corridor. In both images, the contrast between the wall and the floor is quite strong. There is also a black stripe between the wall and the floor which helps to further emphasis the boundaries between the corridor floor and walls. *5.3.1 Corridor Recognition* discusses detection of such boundaries.

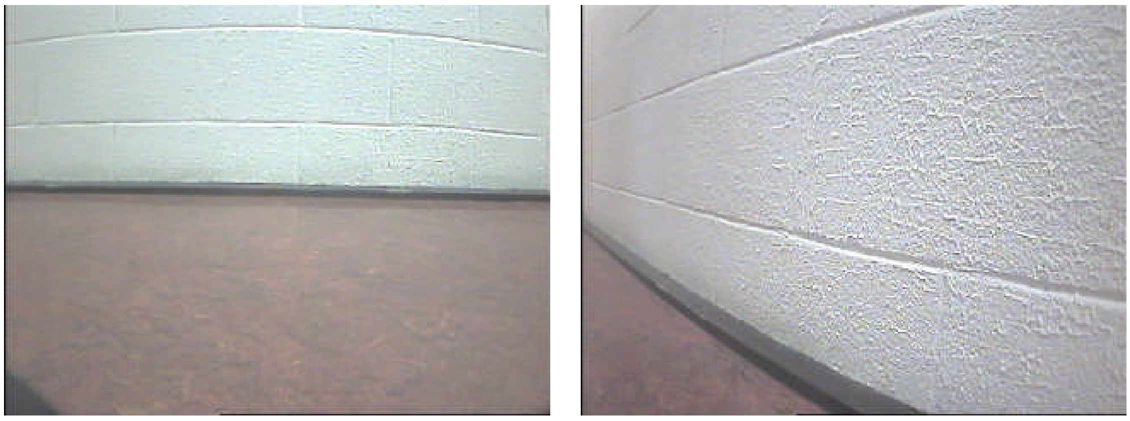


Figure 1. 2 Contrast between floor and wall
(a) with robot 'looking' across the corridor (b) with robot at an angle to the corridor

1.6.2 Varying Light Conditions

Lighting conditions in the corridors vary depending on the time of day, and on whether the electric lights are on or off. The images in Figure 1.3 illustrate the effect of changes in illumination in a particular corridor. Figure 1.3a shows the corridor with the light off and Figure 1.3b shows the same corridor with the lights on.

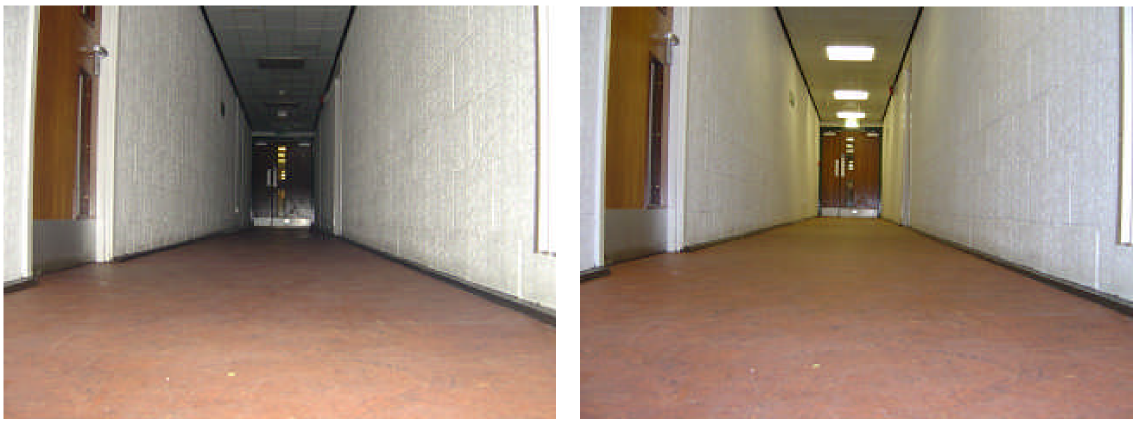


Figure 1. 3 Effects of lighting
(a) A corridor scene with electric lights switched off (**b)** Same scene with lights switched on

In the case of this example, the lights introduce addition strong features as shown in Figure 1.3b, such as the edges of the electric lamps which are not important for navigational purpose, but will consume additional processing resources. The analytically designed system, the Hough transform based vision system, developed in this work is able to cope with this. (The artificial neural network based systems are not designed in detail, as is usual with such systems. They are just trained with examples of inputs and required outputs).

1.6.3 Reflections

Strong reflections such as reflections of electric lights alter images - significantly in some cases. They introduce strong features which are not helpful for navigational purpose, and in some cases are capable of confusing the robot. Figure 1.4 shows some examples of the effects of reflection. Some steps were included in algorithms for processing images to enable the Hough transform based system developed to cope with the effect of reflections. *5.2.3 Vanishing Point Estimation* contains such steps.

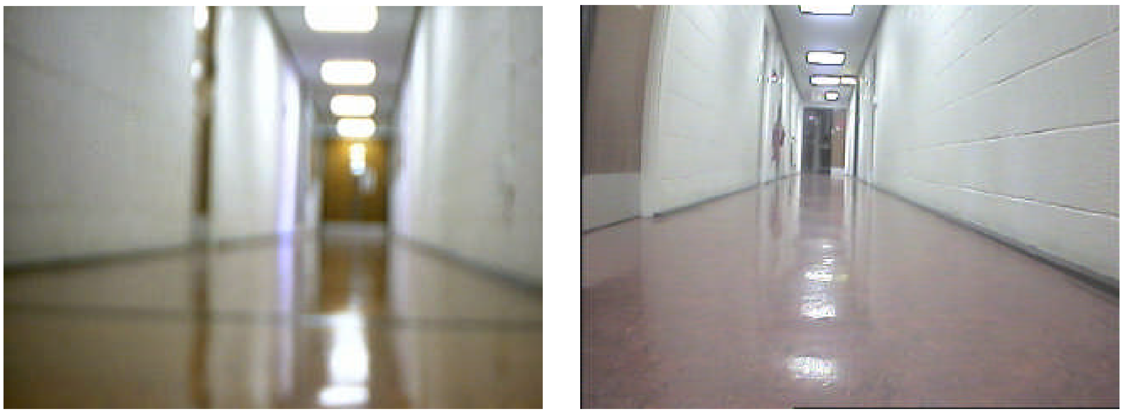


Figure 1. 4 Alterations due to reflection

1.6.4 Doors

In addition to wall/floor boundaries discussed in *1.6.1 Contrast between floor and walls*, another important feature in the kind of environment chosen is doors. Figure 1.5 shows some doors as they appear in the environment. Detection of doors is discussed in more detail in *6.3.2 Door(s) Recognition*.

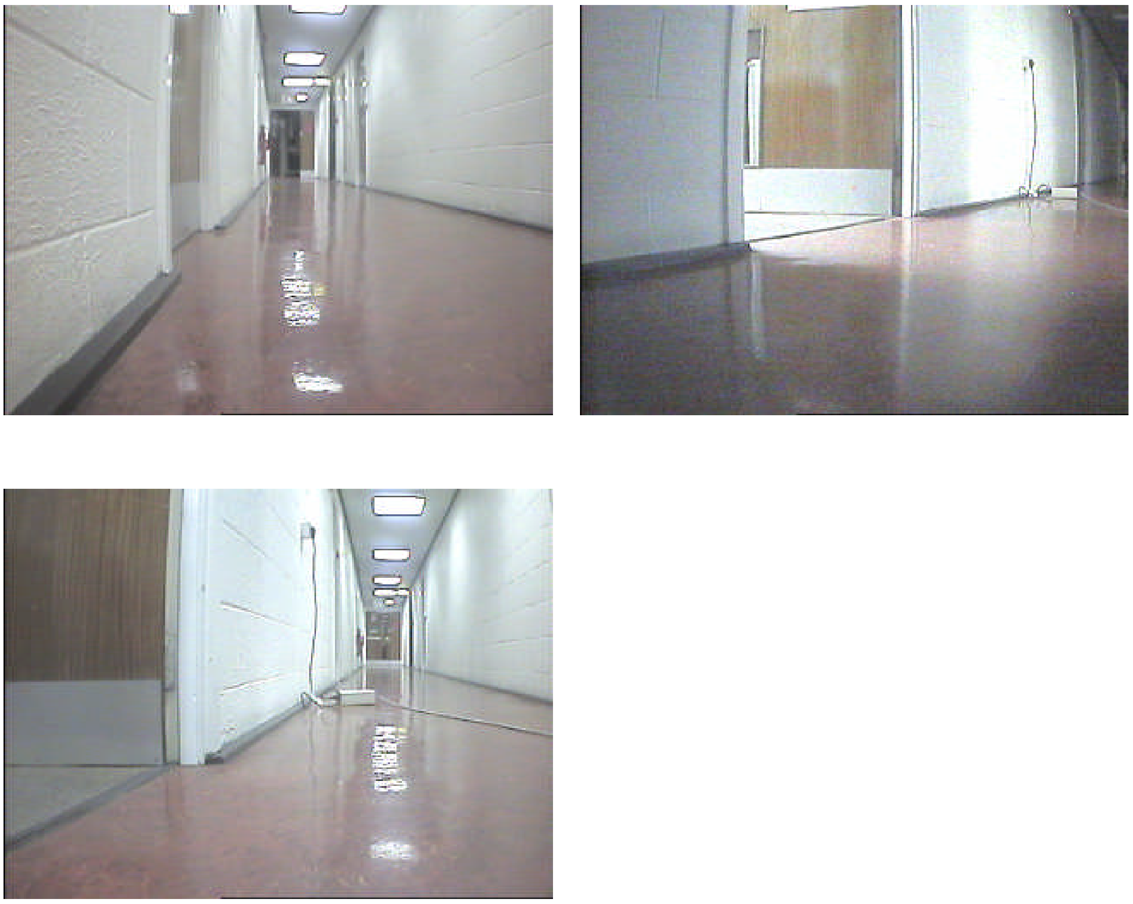


Figure 1. 5 Doors in the corridor environment

1.6.5 Other Objects

Other objects show up in the corridors which can 'confuse' the robot. As an example, the image in Figure 1.6 shows a radiator which can look very much like a door after the image has been processed. This possibility is taken care of during post-processing of the image discussed in *6.3.2.2 Wall-Door Recognition*.

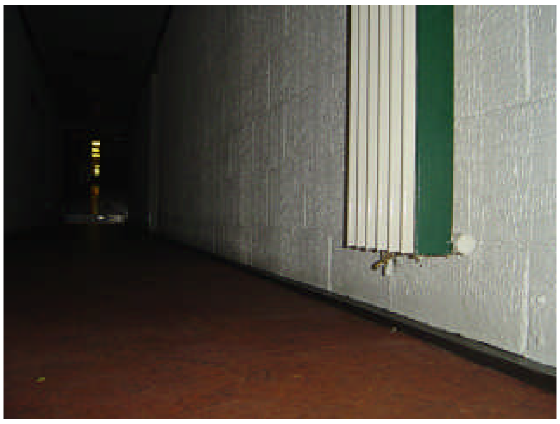


Figure 1. 6 A radiator in the corridor environment

1.7 Project Objectives

The original objectives of this project were to:

1. Develop a mobile robot vision system based on line detection using the Hough transform
2. Develop a mobile robot vision system based on artificial neural networks, which mimics some or all of the stages of the Hough transform based system
3. Compare the computational efficiencies and navigational accuracies of the two systems
4. Look into the feasibility of a new system that is a hybrid of the two, which will draw from the strengths of both systems
5. Document the computational efficiency and navigational accuracy of the new hybrid system

As the work progressed, pursuance of objectives 1 and 2 took much more time than originally anticipated. Necessities to do investigations evolved that were not foreseen. This made it impossible to meaningfully pursue objectives 3 and 5 within the time available for this work. The working objectives by the end of the project are listed in *Chapter 8 Summary, Conclusions and Suggestions for Future Work*.

1.8 New Ideas Developed in this Work

A number of new ideas have been contributed by this work. Some of them are summarised in this section.

1.8.1 Modified Thinning Algorithm

(Park and Chen 2000) present a method of thinning that is an improvement on the most popular methods. However, step one of their method was found to be unsuitable for applications such as the ones in this work. A modification is proposed for that step. Further details are presented in *4.3 Edge-Thinning*.

1.8.2 Effects of Thinning

This work has done an experiment to investigate the effect that thinning of the edge image subsequently has on the quality of results and time taken for the Hough transform and related post-processing. This is discussed in *4.4 Effect of Thinning on Hough Transform Processing Time and Results*.

1.8.3 Automatic Determination of Threshold for Peak Detection

A method has been developed for automatic determination of threshold for peak detection. Rather than use a fixed threshold for every image, this method works out the most suitable threshold to use for every image based on a target number of lines to expect. This target number of lines is worked out based on empirical results. This method is discussed in detail in *5.1.6 Peak Detection*.

1.8.4 Peak Detection Scheme

This work has proposed a peak detection scheme which is centred around the use of the butterfly filter, but cuts down on the amount of arithmetic processing which characterises the use of the filter, by carrying out peak detection by threshold application before applying the filter only to accumulator array points that have been returned as peaks. The scheme is discussed in *5.1.6 Peak detection*.

1.8.5 Actual-Lines Extraction Scheme

The Hough transform finds parametric lines of infinite length. Various schemes have been proposed for determining actual lines on the image from the infinite ones, but mostly have one problem or the other. This work proposes a method which is an improvement on many of the other schemes. It tracks points which contributed to the accumulation for a peak in the accumulator array (which corresponds to a line in the original image), and from them finds valid sub-lines of the parametric line. It then determines endpoints and length for each sub-line. This method is discussed in detail in *5.2.1 Determination of Actual Lines.*

1.8.6 Line Categorisation Scheme

This work has developed a scheme for categorising line segments, which can be helpful for navigation-feature recognition. This is presented in *5.2.4 Type and Position Assignment to Lines.*

1.8.7 Features Recognition Schemes

This work has developed various schemes for recognition of high level features such as corridors and doors from lines found with the Hough transform. The schemes are discussed in *5.3 High Level Features Determination*.

1.8.8 Navigation Algorithms

This work has proposed algorithms to enable a mobile robot to follow simple navigation programmes taking into account high level features detected. These are discussed in *5.4 Navigation*.

1.8.9 Neural Network Scheme for Detecting Lines

This work has developed neural networks to detect lines in thinned edge images by breaking down the image into sub-images. These are presented in *5.2 Vision for Navigation based on Sub-Images Processing.*

1.8.10 Hybrid Vision Systems

This work has investigated the feasibility of some Hough transform/Neural Networks hybrid vision systems, and developed one of them. These are discussed in *Chapter 7 Hybrid Hough Transform/Neural Networks Vision Systems for a Mobile Robot*.

1.9 Outline of Other Chapters

Chapter 2 Background gives further background to the project by explaining what the Hough transform and Artificial Neural Networks are in a bit more detail than was done by this chapter.

Chapter 3 Related Work reviews some previous work related to various aspects of this work in the literature.

Chapter 4 Pre-Processing discusses the pre-processing employed by the current work.

Chapter 5 A Hough transform Vision System for a Mobile Robot explains how the Hough transform was used in this work as the core of a vision system for a mobile robot. It explains how results from the Hough transform are processed to detect navigationally important features in the environment such as corridors and doors, and proposes some algorithms to enable robot navigation under the control of human-type navigation programmes for the robot based on features found.

Chapter 6 Mobile Robot Vision Systems based on Line Detection using Artificial Neural Networks explains the plan for two other vision systems based on Artificial Neural Network techniques. Not the entire plan was implemented successfully. The success (and lack of it) achieved are presented.

Chapter 7 Hybrid Hough Transform/Neural Networks Vision Systems for a Mobile Robot describes efforts to develop another vision system by hybridising the systems described in Chapters 5 and 6.

In *Chapter 8 Summary, Conclusions and Suggestions for Future Work,* this thesis is summarised with major achievements and conclusions highlighted. Suggestions are made on how this work could be progressed.

Chapter 2 Background

This chapter further introduces the project by explaining in general terms what the Hough transform (HT) and artificial neural networks (ANNs) are in more detail than the introduction in *1.1 Project Title and Key Concepts*.

Details of the implementation of the HT for this project are presented in *Chapter 5 A Hough Transform Vision System for a Mobile Robot*, and details of the use of ANNs are presented in *Chapter 6 Mobile Robot Vision Systems based on Line Detection using Artificial Neural Networks*.

Chapter 1 Introduction already discussed the strengths and weaknesses of both systems and explained the motivation for a hybrid of the two. Hybrid systems are discussed in *Chapter 7 A hybrid Hough transform/neural networks vision system for a mobile robot*.

2.1 The Hough Transform

The Hough transform is an image processing technique invented to enable automatic recognition of lines (Hough 1962). It has since been applied to detection of other kinds of regular shapes and even non-regular shapes in images (Low 1991). It has the effect of reducing the search for features such as lines in the original image to a search for a point in a new transformed image, where curves intersect (Djekoune and Achour 2000). The Hough transform is only used for detecting straight lines in this work, therefore versions for detecting other features are not discussed any further.

The sections which follow give a broad overview of the Hough transform. In *Chapter 5 A Hough Transform Vision System for a Mobile Robot*, the use of the transform in this work is discussed in further detail.

2.1.1 Input Images for the Hough Transform

Input images for the Hough transform can be robot-mounted-camera images of the surroundings of a mobile robot as will be the case with this project, and as is the case with other works such as (Djekoune and Achour 2000), and (Espinosa and Perkowski 1991) to name a few. However, they can also be of one of several other types of input data such as sonar data used in (Yun et al 1998) and (Banjanovic-Mehmedovic et al 2004) for example, and range-measuring laser used in (Forsberg et al 1995). (Hough 1962) in which the transform was used originally, had pictures of bubbles in a bubble chamber as input.

Visual images are not necessarily the best kind of input for the Hough transform. The Hough transform of range data is ‘cleaner’ according to (Forsberg et al 1995). Also range data can give an orthogonal view of an environment whereas visual images will normally be two-dimensional and so features such as lines will be perspective impressions rather than what they would be actually (in three dimensions).

This project uses visual images as input as the purpose of the project is to implement vision systems.

2.1.2 Pre-Processing

Pre-processing is commonly performed in image processing to achieve a new image from an input image, suitable for application of so-called middle-level image processing techniques such as the Hough transform. Pre-processing for visual images can include such processes as resizing, histogram equalisation, edge-detection, brightness adjustment, etc.

In this work, at least two major results are achieved from pre-processing. One is that the image is resized to a size at which application of the Hough transform is feasible, while sufficient detail is maintained for the image to be useful. This is necessary because the Hough transform can be very demanding on computing resources, and the larger images are, the more computer time and memory are consumed by the transform. If images are too large, the time required can make application of the Hough transform so time consuming that it is not useful for real time purposes such as mobile robot navigation (Dempsey and McVey 1992).

The second important result from pre-processing is the edge image – a black and white image in which black pixels denote edges in the original image. Such black pixels will be referred to as edge pixels in this work.

There are two broad approaches to edge-detection (Leavers 1992). In the first approach, template matching, the image is convolved with templates of the edges under detection. In the second approach, the differential gradient approach, local intensity gradients are estimated by convolution with suitable filters or masks. Two filters, one for the x direction and one for the y direction are needed. This approach is more computationally intensive than the template matching methods, but gives more accurate results.

The differential gradient approach is used in this work with the Sobel edge-detection filters. They are the most commonly used differential gradient filters (Leavers 1992). They do fail to detect edges which are very gradual. For such edges, the Laplacian edge filters, for example, are more appropriate (Low 1991). The Laplacian filters, however, do not say the angle of the edge, which the Sobel filters do, and this is important for the thinning stage of pre-processing discussed shortly in *4.3 Edge Thinning*. Moreover, the nature of the images in this work is such that edges are sharp enough to be picked up by the Sobel filters, and results from using the Sobel filters have been satisfactory.

The edge-image obtained from Sobel edge-detection is usually unnecessarily thick and would make application of the Hough transform much more computer time consuming than it needs to be. It is further processed in this work with a thinning algorithm that is a slight modification to one developed by (Park and Chen 2000). Although edge thinning is very helpful, it is not absolutely necessary. (Vaughn and Arkin 1990) for example, just used images straight after Sobel edge detection without thinning.

Characteristics of an image other than its edges can be used as input for the Hough transform. This is sometimes desirable because the number of edge points in an image is usually very high. (Vaugh and Arkin 1990) suggest that corners or even lines could be used and would be faster. They are quick to add though, that edge points give higher tolerance of variations (including partial occlusion) of the objects within the image. As already pointed out, edge points are used in this work as input to the Hough transform.

Pre-processing employed in this project is discussed in detail in *Chapter 3 Pre-Processing*.

2.1.3 Transformation from Image Space to Hough Space

Although (Hough 1962) originally proposed transformation using the gradient-intercept form of the equation of a straight line, the normal (or polar) form of the equation of a straight line has become more popular. *2.1.3.1 Straight Line Hough Transform Using the Gradient-Intercept Form of the Equation of a Straight Line* and *2.1.3.2 Straight Line Hough Transform Using the Normal Form of the Equation of a Straight Line* which follow introduce the two approaches. *2.1.3.3 Resolution* introduces the idea of resolution for the Hough transform.

2.1.3.1 Straight Line Hough Transform Using the Gradient-Intercept Form of the Equation of a Straight Line

Hough's original straight line detecting transform aims at finding out the likelihood that a point (x,y) lies on a line

$$y = mx + c \qquad . \qquad . \qquad . \qquad . \qquad \text{Equation 2.1}$$

in the $x-y$ plane whose length is significant enough for the line to be taken as an important feature in the original image. In equation 2.1, m is the gradient of the line, c is the y-intercept of the line and x and y are standard coordinates of edge pixels.

The transform is based on the principle that every point (x,y) on the line given by equation 2.1, defines a set of m values and corresponding c values. This definition is achieved by rearranging equation 2.1 in the form

$$c = -xm + y \qquad . \qquad . \qquad . \qquad . \qquad \text{Equation 2.2}$$

For a given point, x and y are constant, and so for varying values of m, equation 2.2 is a straight line in another plane – the $m-c$ plane. (Davies 1990) refers to this as point-line duality. The line so derived is the transform of the point.

When using equation 2.2 as a basis for a transform scheme, a problem arises if the line in question is vertical or near vertical. The values of m and c both become infinity for vertical lines and are very large for lines that are nearly vertical. Computer modelling of parameter space to accommodate the plots of m against c becomes impractical because of these large values. Various ways have been proposed for getting around this problem. One way, according to (Low 1991), is to use two sets of plots for parameter space. The standard approach already discussed, using equation 2.2, is used for when $|m|$ is less than or equal to 1.0. When $|m|$ is greater than 1.0

$$x = m'y + c' \quad . \quad . \quad . \qquad \text{Equation 2.3}$$

is used where

$$m' = 1/m \quad . \quad . \quad . \qquad \text{Equation 2.4}$$

and

$$c' = -c \quad . \quad . \quad . \qquad \text{Equation 2.5}$$

This has the drawback of requiring much more memory, and a little additional work is required in putting together the findings of the two accumulator arrays.

2.1.3.2 Straight Line Hough Transform Using the Normal Form of the Equation of a Straight Line

Another way to get around the problem that has become popular is to use the polar form (also called the normal form) of the equation of a straight line

$$\rho = x\cos\theta + y\sin\theta \quad . \quad . \quad . \qquad \text{Equation 2.6}$$

This approach was first proposed by (Duda and Hart 1972). It is based on the principle that for every point (x, y) in the x-y plane, there is a curve defined by equation 1.6 in the $\theta - \rho$ lane. Both ρ and θ are bounded within reasonable limits.

θ is bounded by 0° (inclusively) and 180° (exclusively), and ρ is bounded by -$\sqrt{(\mathrm{h}^2 + w^2)}$ (inclusively) and $\sqrt{(\mathrm{h}^2 + w^2)}$ (exclusively), where h, the highest magnitude of height on the image with the origin at the centre of the image, is given by $h = \lceil H/2 \rceil$, H being the height of the image, and w, the highest magnitude of width on the image with the origin at the centre of the image, is given by $w = \lceil W/2 \rceil$, W being the width of the image.

This approach is employed in this work and is the basis for any further discussion on the Hough transform. Implementation of this approach for this work is discussed in detail in *Chapter 5 A Hough Transform Vision System for a Mobile Robot*. The gradient-intercept $(m-c)$ based transform is not discussed any further.

2.1.3.3 Image Space and Parameter Space

This input image for the Hough transform has points in the $x-y$ plane and is commonly referred to as image space. The new image has points in the $\theta-\rho$ plane and is sometimes called Hough space after Paul Hough the inventor of the technique. The qualities m and c are commonly referred to as parameters of the transform, and Hough space is for that reason also sometimes referred to as parameter space. Hough himself referred to his $m-c$ plane as the transformed plane in his original application for a patent (Hough 1962). In this work, the expression parameter space will be used because it appears to have become the most common name for the new plane.

2.1.3.4 Parameter Quantisation Intervals

Equation 1.6 is used to transform every edge pixel (x,y) in image space. The intervals at which the parameters θ and ρ are recorded can significantly affect the quality of the results of the transform and the processing time and space required.

Usually, x and y are examined at intervals of 1 pixel. That makes it natural to select an interval of 1 pixel for ρ, the distance parameter. ρ is evaluated as a floating point number using equation 1.6, so the result is quantised to the nearest whole number.

The choice of interval for θ is not as straightforward. To determine the transform for each edge pixel, various values of θ are taken at a regular interval. Generally speaking, the smaller the interval used, the better the results obtained but also the higher the computing time and memory requirements. However, (Leavers 1992) has demonstrated that in detecting a line of length L, if the θ interval goes below $\tan^{-1}(1/L)$, a number of peaks result along the θ direction instead of a single peak, and this can lead to the transform yielding several lines where there was only one. Depending on the extent of the spread, it might be possible to consolidate the multiple peaks back to a single peak, eliminating false ones. Consolidation of peaks is discussed further in *5.1.6 Peak Detection.*

The choice of interval for θ for the current project is discussed in *5.1.2 θ Resolution*.

2.1.4 The Accumulator Array

Closely associated with parameter space is the accumulator array. It is a two dimensional array superimposed over parameter space. The actual number of elements in the array depends on the level of accuracy required from the application in question. For the highest resolution, there will be an element of the accumulator array for every point (θ, ρ) in parameter space, and the value of the element will simply be the value of the accumulation at the corresponding parameter space point (θ, ρ). How the value of accumulation is derived is discussed shortly in *2.1.5 Accumulation*. For this resolution, the accumulator array will have as many columns as the range of θ, and as many rows as the range of ρ. This resolution is used in this work.

For other resolutions, the parameter space is divided into cells that are usually of equal sizes. It is common to refer to these cells as bins. The accumulator array is set up to have elements corresponding to each of these cells. Entries to the array are aggregate sums of the values of accumulation for parameter space points within the cell corresponding to the particular accumulator array entry.

The resolution of the accumulator array is chosen bearing in mind processing time and space requirements on the one hand and the level of accuracy required for feature detection on the other. The amount of processing time and memory required both vary linearly with the resolution chosen (Atiquzzaman 1992).

2.1.5 Accumulation

The accumulator array is usually set up as a zero array initially. Individual entries are then increased in the course of application of the transform. The value of each entry is called the accumulation of that entry. How this increase is done varies. The most popular way is to increase the value of each entry by 1 whenever a curve resulting from a transform crosses it. This approach is use in this work.

(Low 1991) has suggested that instead of increasing just the appropriate accumulator array point, a square block of points with the point (θ, ρ) as centre is incremented. If a block 3 x 3 points in size is used for example then 9 bins are incremented for every $\theta - \rho$ pair. This results in thick parameter space lines and can help detection of intersection points during peak detection.

Three ways have been suggested by (Low 1991) for further enhancement of peak detection:

1. Addition of the gradient magnitude of the edge for accumulation rather than simply incrementing by 1. This plots a measure of the likelihood of the source point being an edge.
2. Only plotting for θ values within a reasonable distance of the gradient direction of the edge.
3. Incrementing a 3 x 3 block of bins but incrementing the middle entry with a larger figure than the others.

2.1.6 Peak detection

When all edge pixels have been processed, accumulator array entries will have varying values. Those that have not been affected by the transform for any edge pixel will still have their initial value of 0. All others will have positive integer values equal to the number of times a transform curve has crossed them. Some accumulator array entries will have higher values than their neighbours and are referred to as peaks. Peak detection is the process of determining which bins are peaks.

Peak detection is achieved by application of a threshold. All accumulator array entries above a certain threshold automatically qualify as peaks and all others are not peaks.

Depending on the application, thresholds can be fixed or they can vary for every image.

2.1.6.1 Fixed Threshold

In applications where it is known how many pixels a line (or whatever feature is being detected) needs to have to be significant, it is possible to determine and use a constant threshold.

Use of a fixed threshold is advantageous in being very easy to implement, and requiring no significant additional computing time as would be required by constantly evaluating the threshold. A fixed threshold can be effective where target features do not vary widely.

It has the drawback of not being sensitive to the nature of specific images. A busy image would tend to have much higher accumulations and using a fixed threshold would mean that such an image returns many more lines than are sensible. These can make further processing more complicated than it needs to be. A sparse input image would return fewer lines than are required to sensibly interpret the image even though there is enough information in the image to find those lines with a lower threshold.

2.1.6.2 Variable Threshold

Where it is not practical to use a fixed threshold, different thresholds would need to be determined for every image processed. A few approaches can be taken. One proposed by this work is to determine threshold based on an approximate number of expected outcomes. In summary, this means that a decision is taken on how many significant elements are required, n say, and then aiming at selecting the top n elements of the array. This is discussed in detail in *5.1.6 Peak Detection.*

It is also possible to determine the appropriate threshold by introducing a number of different ones on a trial and error basis. The outcome of the transform with each threshold can be studied and then the threshold with the best outcome can be maintained. This approach requires a lot of manual study of results, in addition to understanding of the quality of the outcomes. It is therefore not suitable for real time applications.

2.1.6.3 False Peaks

In most situations, peaks are not easily distinguishable. This can be due to a number of possible situations. An example is a situation where several accumulator array entries in a neighbourhood have values higher than the edge detection threshold. Although one of them is higher than all others, several peaks which are not actual peaks are returned by the application of the threshold. This work refers to them as false peaks.

Where they exist, elimination or at least minimisation of false peaks is necessary. One method proposed by (Boyce et al 1987) is the use of the butterfly filter. They worked out a 5 x 9 convolution filter which when applied to an accumulator array will emphasis actual peaks while suppressing false peaks. They determined the filter based on the fact that accumulator array entries due to points on a straight line have a characteristic shape that resembles the shape of a butterfly, hence the name. The filter was determined such that a maximum positive response is obtained when it coincides with the butterfly shape due to a line, and a zero response is obtained in a uniform area of the array.

Because application of the 5 x 9 butterfly filter is computing time demanding, (Boyce et al 1987) also developed what they call a reduced butterfly filter, which is 3 x 3 in size. It has a similar effect to the full filter although it does not take as many points into accounts, and so may not be as effective. It drastically reduces the amount of arithmetic required to evaluate convolutions with the filter however. This reduced butterfly filter is employed by this work. The butterfly filter is discussed further in *2.1.2 (Boyce et al 1987)* and its implementation in this work is discussed in further detail in *5.1.6.2 Application of the Butterfly Filter.*

2.1.8 Reconstruction

When peaks have been correctly determined, the parameters which represent them can be used as representations of the lines found. However, the transform yields no information about the length of the lines or their endpoints. (Leavers 1992) describes this as one of the disadvantages of the method. *2.1.8.1 Line Equation Reconstruction* discusses the reconstruction of the gradient-intercept form of the equations of the lines found. This is not absolutely necessary but may be helpful if a graphic illustration of the line is required, or as a step towards determining endpoints and length using some techniques. *2.1.8.2 Endpoints and Length Determination* introduces the determination of endpoints and lengths for the lines found.

2.1.8.1 Line Equation Reconstruction

Peaks detected in parameter space are reverse-transformed to lines in image space. In this work, as the transformation was done with equation 1.4, reversal can be achieved by re-arranging equation 1.4 in the form of equation 2.1 to give

$$y = \rho \operatorname{cosec}\theta - x\cot\theta \qquad . \qquad . \qquad . \qquad \text{Equation 2.7}$$

This implies that

$$m = (-\cot\theta)\,, \qquad . \qquad . \qquad . \qquad \text{Equation 2.8}$$

and

$$c = \operatorname{cosec}\theta \qquad . \qquad . \qquad . \qquad \text{Equation 2.9}$$

2.1.8.2 Endpoints and Length Determination

The lengths of lines found are not determined from the Hough transform. Several methods can be used for estimating the length and/or end-points of a line. This work keeps a record of all the points that contributed to the accumulation of each point in the accumulator array, so that for any point returned as a peak, all the points on the line are known. They are then checked against certain criteria to determine if they make-up valid sub-lines for the line. Sub-lines and valid sub-lines are defined in *5.2.1 Determination of Actual Lines*. Endpoints of a sub-line are then determined as the points farthest away on both ends of a valid sub-line along the direction of the line, and its length is the distance between its endpoints. With this approach, more than one sub-line can be found on a line, and this takes care of situations in which lines along the same direction are returned as one line – a major problem of the Hough transform (Leavers 1992). This approach is detailed in *5.2.1.2 Estimating Valid Sub-Lines*.

(Leavers 1992) suggests another method. A search corridor of width of 5 pixels is set up in the edge image with the line defined by equation 1.7 as the centre of the corridor. Coordinates of edge points found within this corridor are stored and the points are deleted from the edge image. If points found do not account for up to 50% of the accumulation corresponding to that line, the search corridor is rotated by angles up to the error in θ. Edge points previously deleted are restored and the search is started all over again. A few problems with this approach are:

(1) If an edge pixel has contributed to the accumulation for more than one line, it is only accounted for once and deleted, and there is the possibility that points which should genuinely be on a line are not reported to be on it. This can affect the accuracy of the endpoints and lengths returned eventually. (Leavers 1992) propose to search for points in descending order of the transform strengths of line segments as a way to minimise this effect, but even this proposal does not eliminate the possibility.

(2) A threshold of 50% of points to be found means that up to 50% of points on some line segments may not be included in the process of determining the endpoints and lengths for those line segments, and there is potential for error as the ones not included may be the endpoints.

(Low 1991) suggests other approaches:

1. application of corner detection methods and estimating line lengths from corners
2. setting up of four further accumulator arrays to hold the x and y values of the most north-easterly and the most south-westerly ends of the lines at the time of transformation

3. setting up four further accumulator arrays, storing values for wx, wy, $(wx)^2$ and $(wy)^2$ and working out ends of line using $\frac{\sum wx}{\sum w} \pm 2 \times \sqrt{\frac{\sum (wx)^2}{\sum w} - \left(\frac{\sum wx}{\sum w}\right)^2}$ for the x coordinates and $\frac{\sum wy}{\sum w} \pm 2 \times \sqrt{\frac{\sum (wy)^2}{\sum w} - \left(\frac{\sum wy}{\sum w}\right)^2}$ for the y coordinates.

   (w is the value with which the main accumulator array is incremented for each (x, y))

Problems with the first suggestion include the complications and computing overheads related to corner detection. The second suggestion is the easiest to implement but has the drawback that determination of end points for lines with a northwest – southeast orientation does not succeed.

The third suggestion assumes that the points are spread evenly across the line, but this is often not the case, and the results can be very misleading.

This work has developed another approach to determination of endpoints and length as mentioned earlier in this section. It is discussed in *5.2.1 Determination of Actual Lines*.

2.2 Artificial Neural Networks

The discussion in this section is based on discussions in (Orr and Schraudolph 1999), (Jagadeesan 2006), and (Anonymous).

Artificial neural networks (ANNs) are commonly used in artificial intelligence systems to enable machines to learn to recognise and classify input patterns from a few representative samples (Picton 2000). They are also called simulated neural networks (SNN) or just neural networks (NN). They consist of mathematical models called artificial neurons designed to take in numeric input corresponding to some input pattern, process these using some mathematical procedure and give outputs depending on input values in a manner imitating the functioning of biological neurons.

The use of ANNs is growing rapidly in such areas as robotics, visual pattern recognition, speech pattern recognition and research into the human brain-mind process (Harvey 1994).

Artificial neurons and neural networks have been implemented in hardware as electronic circuits, often integrated on VLSI chips. It is also possible to implement ANNs as software simulations in reasonable time. This has the advantage of being much more flexible, and portable, and less expensive.

This section briefly explains what ANNs are in general terms, and then gives a brief introduction to back-propagation networks – the specific kind of ANNs used in this work. The first sub-section briefly introduces biological neural networks which inspired artificial neural networks and which artificial neural networks were designed to model.

2.2.1 A Quick Look at Biological Neurons and Neural Networks

Sensation and reasoning in animals is possible because of the existence of networks of nerve cells or neurons within them. In higher animals, billions of neurons constitute neural networks.

This network is capable of performing complex communication and information processing within the animal and between it and its environment. The main task of the nervous system is to ensure that the organism reacts optimally to circumstances within it and in its environment.

The nervous system in biology can be divided into the “Central Nervous System” (CNS) which consists of the brain and the spinal cord, and the “Peripheral Nervous System” (PNS) which consists of the sensory and motor neurons, and connects the outside world to the CNS.

Sensory neurons receive stimuli from the outside world and transmit them to the central nervous system in the form of nerve impulses. The CNS processes the stimuli and passes control information to motor neurons which carry out the body's actions via muscles and glands.

Figure 2.1 illustrates a typical neuron

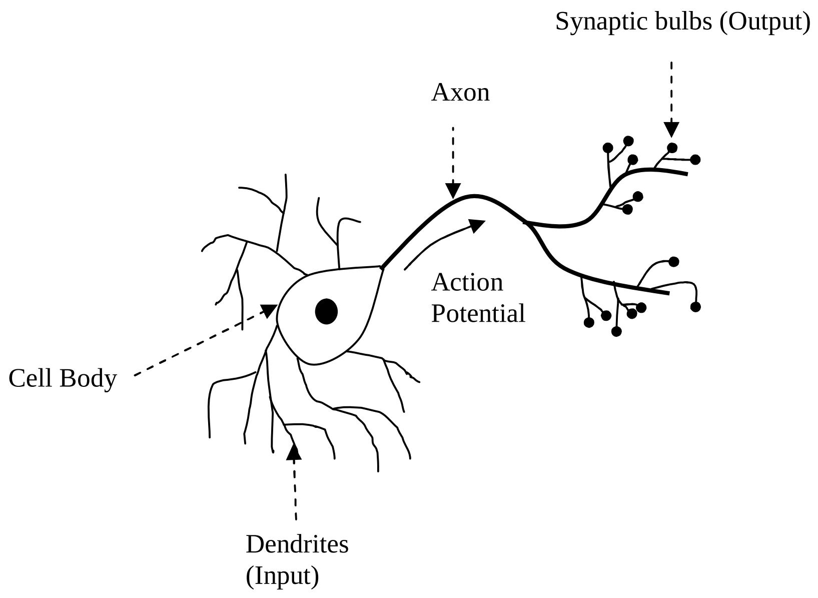


Figure 2.1 A biological neuron
(Source: (Jagadeesan 2006))

At the centre of a neuron is the cell body or soma. Information processing takes place within the cell body. Root-like structures which receive input signals and transmit them to the cell body are called dendrites. Dendrites receive pulse information from other neurons or outside of the body of the animal. When pulse information received from all these dendrites accumulates to a certain level, an ionic chemical reaction takes place resulting in a pulse. This pulse is called an action potential. It is sent out of the neuron through the axons.

Each neuron is connected to many other neurons (in the order of about 10,000 although actual number varies greatly) and the total number of neurons and connections in a network can be very large. The connection between one neuron and other neurons it transmits to is done via its axons and their dendrites, and is commonly called a synapse.

Figure 2.2 illustrates the operation of a synapse. A neuroactive substance called a neurotransmitter effects the transfer of information in the form of electric signals from the transmitting to the receiving neuron. Neurotransmitters are chemicals which are released from the transmitting neuron and which bind to receptors in the receiving neurons. The extent to which the signal from one neuron is passed on to the next depends on many factors characterising the synapse. These include the amount of neurotransmitter available, the number and arrangement of receptors, the amount of neurotransmitter reabsorbed, etc.

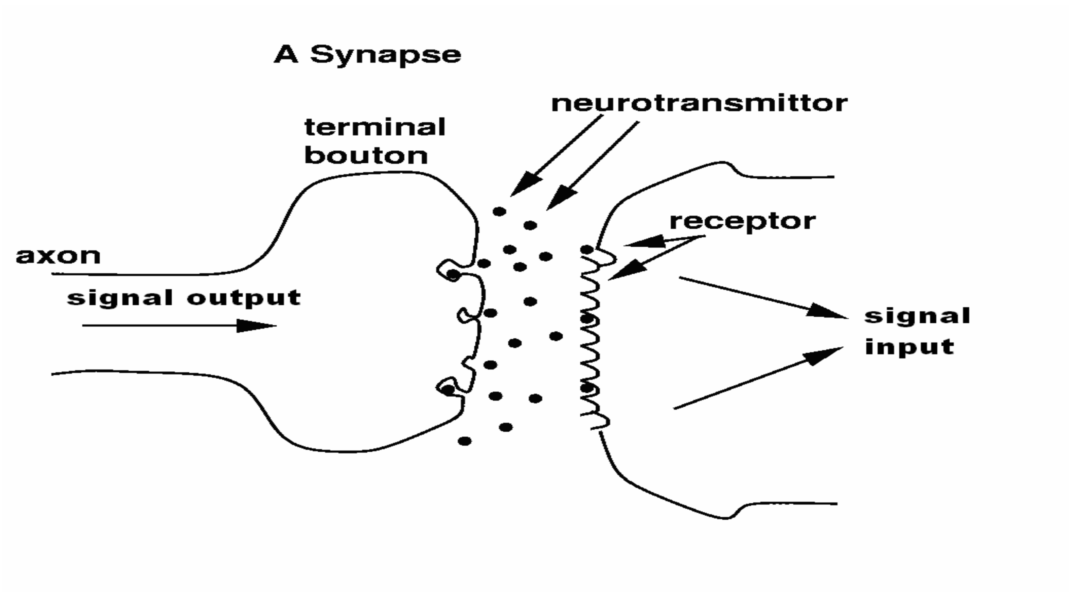


Figure 2.2 A synapse
(Adapted from: (Orr et al 1999))

Learning is achieved by altering the strengths of connections between neurons and by creating connections to other neurons or removing connections to other neurons. Altering the strength of a connection can be achieved by adjusting the amount and type of the neurotransmitter between neurons.

2.2.2 Artificial Neurons and Neural Networks

An artificial neural network (ANN) is an interconnected group of artificial neurons set up to employ a mathematical model of biological neural networks for information processing.

An artificial neuron, also called a node, a unit, a neuron, a neurode, or a processing element, consists of simulations of one or more input links (imitating a dendrite), an information processing area (representing a cell body), and one or more output links representing axons. It receives input from some other neurons, or from an external source. Each link has an associated weight w, which can be modified so as to model changes to a synapse during learning. To imitate the build up of signals and processing of information in the cell body of a biological neuron, the artificial neuron calculates the weighted sum of its inputs, and then evaluates some function f for the sum.

This “goes out” to other neurons as input for further processing, or out of the system as the output (or part of the output).

The function f is called the activation function (also called the transfer function). The weighted sum of inputs to a neuron is called the net input to the neuron, and is often called net.

As in the biological case, a single neuron is not very useful by itself. Several of these artificial neurons are interconnected to form a useful artificial neural network.

The arrangement of neurons in an ANN is called the topology or the architecture of the network. ANNs do not attempt to model the topology of a biological system exactly as it is much too complicated. Instead, ANNs usually have neurons arranged in a few layers.

2.2.3 Types of Artificial Neural Networks

There are several types of artificial neural networks. They vary according to how their constituent neurons are implemented, their topologies, and their specific purpose among other things. They can broadly be divided into three categories – the feed-forward networks, the unsupervised networks and the Hopfield networks (Sonka and HLavac 1999)

Feed-forward networks are so called because they admit input data from the input nodes, and pass these in one direction towards the output nodes where the answer may be read. There is no connection from the outputs back to the inputs. One of the earliest types of feed-forward networks consisted of networks called perceptrons. They are capable of performing classification and function evaluation tasks in a wide variety of areas. They fail in solving problems that are not linearly separable however. Another type of feed-forward network called the back-propagation network which introduces one or more inner layers is able to deal with linearly inseparable problems. It has become very popular and is the type employed in this work. It is discussed further in *2.2.3 Back-Propagation Network* which follows.

Feed-forward networks generally require to be trained with a set of input data for which correct output is already known using some training algorithm. This teaches the network, so that in live mode, it can generate output for unknown patterns based on generalisation from what it has learned.

Self-organising networks are designed to recognise patterns without any training set for which inputs and corresponding outputs are known, unlike feed-forward networks. The best known types of networks in this class are the Kohonen feature maps, which can be used for clustering because similar input data generate the same output.

Hopfield networks do not have designated input and output nodes; rather, the current configuration of the network represents its state. Neurons are fully interconnected and have discrete outputs which can take values of 0 (or sometimes -1), or 1. Weights are computed initially using known exemplars and do not change subsequently. Hopfield networks are mostly used to solve optimisation problems.

2.2.4 The Back Propagation Network

Back-propagation networks consist of a number of artificial neurons organised into at least three layers. Figure 2.3 illustrates a typical back propagation network.

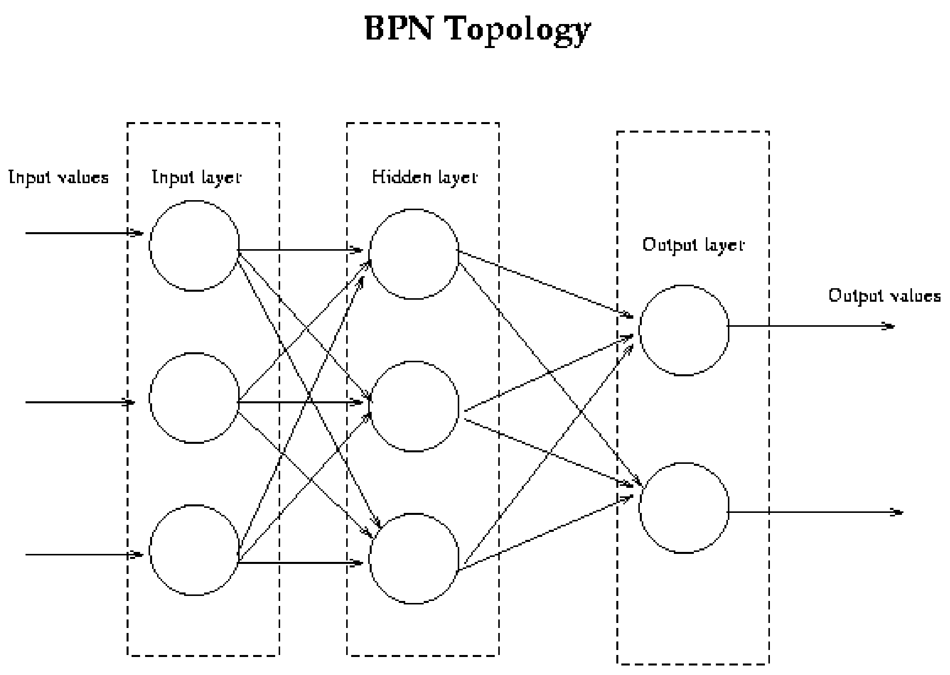


Figure 2.3 Neurons arranged in layers in a typical back propagation network
(Source: (Crochat and Franklin))

A layer referred to as the input layer has neurons equal in number to the number of inputs to the network and each neuron in this layer is fed with one of the inputs to the network when the network is in live mode. The input layer then passes whatever input it has received to one or more inner layers (also called hidden layers) applying weights to them, whose values would have been determined in the course of training of the network. Neurons in the inner layer sum the weighted inputs they receive to obtain a ‘net’ as described in *2.2.2 Artificial Neurons and Neural Networks*, and then pass the value of ‘net’ through a squashing function which limits the range of possibilities of output from these neurons. This project employs the sigmoid function which limits output to a continuous range between 0 and 1, and is the most commonly used squashing function for back propagation neural networks.

Before the network can be put to use, it needs to be trained. Training of a back propagation network is done by presenting a set of inputs and corresponding desired outputs to the network. This set is called the training set. A set of weights for the links in the network would have been chosen randomly initially. The inputs to the network are first received by the input neurons, which passes them on to the inner layers applying the current weights to them. The inner layers compute the weighted sums and apply the squashing function to obtain their outputs. The outputs to the current inner layer are passed on to the next inner layer if there is any, or to the output layer. The output layer performs the last round of information processing and calculates its output which is also the output of the network. The values of the calculated outputs are compared to the values of the desired or target outputs for that input pattern. The differences or errors are then propagated back through the network adjusting the weights of the links. The network type gets its name from this. The next input pattern in the training set is then passed to the network and the errors are again propagated back and the weights adjusted.

Two parameters, the learning rate, and the momentum term are sometimes introduced. The learning rate is introduced to alter the rate at which training is done, which may be helpful if, for example if the rate of learning is deemed to be so fast it could miss optimal solutions. The momentum term is sometimes introduced to “shake things up” a bit. It scatters the progress being made a little bit and could be helpful to prevent the training process to converge to a solution that may be optimal within a small locality, but not on the bigger scheme of things.

This process is repeated several times with all the input patterns in the training set. An epoch is then said to have been completed. Several epochs are usually done until the errors come below a predefined level. The final weights are then saved, and are used when the network is put to use.

It is necessary to have a maximum number of epochs allowable, so that attempts to train do no continue indefinitely. When this maximum number is reached, the process is stopped. Changes can be made such as alteration of the networks topology, or introduction and/or alteration of parameters like the learning rate and momentum term.

2.2.5 Applications of Back Propagation Networks

The use of BPNs is growing rapidly. They are widely used in image processing for recognition of hand-written characters, faces, fingerprints, gaits, etc., and in visual search engines. They are also used for voice recognition, speech production, RADAR signature analysis and stock market prediction.

BPNs are becoming increasingly useful in robotics. Some tasks where BPN are useful include processing of accelerometer system data (used for balance in two-legged robots), processing of voice commands, navigation, vision, etc.

They are used in manufacturing industries for control, and in business, for mortgage decisions, for example.

Chapter 3 Related Work

This chapter discusses some works in the literature, which are related to the current one in one way or another.

3.1 Works Related to the Performance of the Hough Transform Algorithm reviews some works that analyse performance of the Hough transform and attempts that have been made to improve its performance as algorithms.

3.2 The Hough Transform in Vision Systems for Mobile Robot Navigation and *3.3 Artificial Neural Networks in Vision Systems for Mobile Robot Navigation* respectively, discusses some works that have employed the Hough transform and the back propagation neural network in vision systems for navigation in mobile robots.

3.4 Works which Employ a Combination of the Hough Transform and Artificial Neural Networks looks at some works that have used a combination of the Hough transform and artificial neural networks, and *3.5 Other Related Works* looks at some works which do not quite fit into any of the previous categories, but are still considered related to this work.

3.1 Works Related to the Performance of the Hough Transform Algorithm

3.1.1 The Work of Princen and Others

(Princen et al 1994) have proposed a formal quantitative approach for designing Hough transform algorithms. According to them, the Hough transform can be seen as a hypothesis testing method with each set of indices of the parameter array pointing to a sample which tests if the given parameters fit data from the original pre-processed edge image. The performance of the transform can therefore be quantified.

Their discussion is based on the Hough transform for detecting a curve and so differs from the current work from the angle that this work focuses on the Hough transform for detecting straight lines. Their proposal can however be adapted for straight line detecting Hough transforms. The current work does not attempt to adapt the proposal, but notes it as interesting.

3.1.2 The Work of Boyce and Others

According to (Boyce et al 1987) one of the problems of the Hough transform is that it is unreliable due to the occurrence of false maxima. They present a general convolution filter which enhances true maxima and suppresses false ones. The filter takes advantage of the fact the Hough transform of a straight line has a characteristic shape which resembles the shape of a butterfly, and is therefore referred to as the butterfly filter.

The filter is 9 x 5 in size and is shown below:

$$\begin{bmatrix} -0.1034 & -0.1111 & -0.1111 & -0.1111 & -0.1034 \\ -0.0239 & -0.1111 & -0.1111 & -0.1111 & -0.0239 \\ 0.0501 & -0.0542 & -0.1111 & -0.0542 & 0.0501 \\ 0.0515 & 0.1571 & -0.0732 & 0.1571 & 0.0515 \\ 0.0515 & 0.2386 & 0.8130 & 0.2386 & 0.0515 \\ 0.0515 & 0.1571 & -0.0732 & 0.1571 & 0.0515 \\ 0.0501 & -0.0542 & -0.1111 & -0.0542 & 0.0501 \\ -0.0239 & -0.1111 & -0.1111 & -0.1111 & -0.0239 \\ -0.1034 & -0.1111 & -0.1111 & -0.1111 & -0.1034 \end{bmatrix}$$

Figure 3.1 The butterfly filter

Application of a filter as large as this takes quite a bit of computer time, and therefore may not be suitable for some applications. For this reason, (Boyce et al 1987) worked out a smaller filter, 3 x 3 in size which approximately has the same effect as the 9 x 5 filter and can be implemented much faster. The reduced filter follows:

$$\begin{bmatrix} -1 & -4 & -1 \\ 2 & 8 & 2 \\ -1 & -4 & -1 \end{bmatrix}$$

Figure 3.2 The reduced butterfly filter

There are a few problems with the butterfly filter:

(1) It takes quite a bit of arithmetic to work out the scores for every point of the accumulator array, even with the reduced filter

(2) It still returns multiple peaks within a small neighbourhood of each other in some cases.

The current work goes around these problems by

(1) Applying the butterfly filter to only elements of the accumulator array that have been marked as peaks from application of a threshold to elements of the array after the Hough transform has been implemented.

(2) Selecting only maxima that are higher than all other entries within a 5 x 5 neighbourhood after the filter has been applied.

3.1.3 The Work of Grimson and Others

(Grimson 1990) provide a theoretical analysis of the generalised Hough transformations deriving bounds on the set of accumulations in parameter space in the presence of noise and occlusion in the image, and bounds on the likelihood of false peaks as a function of noise, and occlusion.

They conclude that application of the Hough transforms can be risky as the proportion of false peaks can be very high. They suggest that peaks be subjected to further scrutiny for added confidence.

The subject of their discussion, the generalised Hough transform, is used to find objects of an arbitrary shape. The straight line Hough transform used in the current work is a special case of the generalised transform.

The current work has taken some measures to deal with false peaks. These include employment of a number of processes in peak detection (*5.1.6 Peak Detection*), dismissing of lines found as false peaks if on closer look at the set of points which contributed to them, there is no strong support for the existence of reasonably sized sub-lines within them (*5.2.1 Determination of Actual Lines*), and development of systems which finds sub-lines in sub-images which make up the original image and therefore validating the existence of a peak only if there is evidence for the existence of valid sub-lines in the line it corresponds to (7*.2.2*

Hybrid 5: Use of ANNs to find lines in Sub-Images followed by use of Hough transform to establish 'full-picture').

3.2 The Hough Transform in Vision Systems for Mobile Robot Navigation

3.2.1 The Work of Tyson

Some related work has also been done here at the Robert Gordon University, Aberdeen. One of these, (Tyson 1995), has produced a minimal but fully operational robot video-guidance system based on line detection using the Hough transform. This system is capable of detecting and following a white line on the floor, which is captured by means of a CCD camera. He, however, reports that the time taken to process images makes it simply infeasible for real-time robot control.

The current work investigates the feasibility of improving on the performance of the Hough transform as a basis for vision for a mobile robot, by hybridising it with artificial neural networks.

3.2.2 The Work of Djekoune and Achour

The work of (Djekoune and Achour 2000) aims to navigate a robot down the middle of a corridor. It uses the Hough transform to correct drifts from specified paths between fixed stations. The robot is designed for use in structured indoor environments. It is equipped with a single camera as well as odometers, and infrared and ultrasonic sensors.

In the course of navigation the system corrects drifts due to surface roughness and undulation, due to wheel slippage and due to issues relating to the telemetric sensor used. Localisation of the robot can then be achieved by calculation of its displacement relative to a fixed reference point.

The Hough transform is used to extract line segments and a list of the segments (identified by their θ and ρ values) is maintained.

Corridor boundaries are found as line segments satisfying certain conditions including among other conditions:

1. they are always oblique
2. right border's θ value is between 0° and 90°
3. left border's θ value is between 90° and 180°

Their aim of navigating a robot down the middle of a corridor is quite similar to the aim of this work, of getting a robot to self-navigate through rectilinear environments typified by corridors. The current work also attempts to get the robot to recognise doors and navigate into them.

Their use of hardware to implement edge-detection is very different from the software implementations of the Sobel edge-detection algorithm and a modified version of the thinning algorithm of (Park, 2000) employed for binary edge derivation in the current work.

While (Djekoune and Achour 2000) attempt to track the location of the robot by working out its position relative to fixed reference point, correcting drifts in the process, the current work makes no attempts to track the position of the robot. Instead, navigation is dictated by high level descriptive programmes made up of commands such as go straight, take the first turn left, enter door on the right, etc.

The current work employs a slightly more sophisticated approach to determining corridor edges than the approach of (Djekoune and Achour 2000). Points are assigned to lines based on such factors as their distance to the bottom corners of the image, their distance to the vanishing point if it is known, and the category they fall into which is dictated by their θ value. This is discussed in detail in *5.2.5 Corridor(s) Detection*.

3.2.3 The Work of Vaughn and Arkin

(Vaugh and Arkin 1990) have developed a system which enables a robot to locate a docking station within a manufacturing environment when it is within 10 to 20 feet of the station. It navigates to within that distance using other techniques such as ultrasonic sensing.

512 x 512 pixel sized images were captured and reduced to 256 x 256 pixel sizes for computational efficiency. Images captured were typically cluttered with workbenches, chairs conveyor belts, guidance lines for AGVs used in the laboratory, etc.

The Sobel edge-detection operators are used for edge detection. Each edge is typically several points thick.

From the initial position of the robot, four more positions are generated by moving a distance of 1 foot in any direction. Hough transforms are taken at all five positions with votes cast to the same accumulator array and the position of the dock is returned as the position with the highest number of votes for being the position. Measures of uncertainty of the robots position and orientation relative to the docking station are calculated and maintained. This approach simplifies location of the dock although it means that range and orientation information cannot be recovered.

With the algorithm, it takes about 10 minutes to run the cluster of Hough transforms on a MicroVAX II – nowhere near real time. The typical image has the order of 10,000 edge points (after thresholding) and the model has about 700 edge points. This means approximately 7 million votes are being cast for each run of the Hough transform - 35 million for the entire cluster.

The algorithm could be much faster with parallel computing. Also, using lines or corners for features instead of edges would take less time as the model would be composed of only a few dozen features with only several hundred lines or corners being extracted from the image (resulting in only several thousand votes).

Their work uses visual image as input for the Hough transform with the purpose of generating control information for a mobile robot and to that extent is similar to the current work. There are a few differences however. One that is easily noticeable is that their system uses other navigation techniques while the current work relies entirely on visual input.

They resize images to a 256 x 256 size which is considerably larger than the 128 x 96 size that this work employs. This size would have contributed to the typical processing time they recorded, which is so high it cannot be used for real-time applications. Choice of image size for this work is discussed in *3.1.3 Resizing*.

Another factor that would have contributed to the high processing time would be the use of an edge-image with edges several pixels thick. The current work implements a thinning algorithm which gives an edge-image with edges only a pixel in thickness. This has been found by this work to reduce the number of edge-points by up to about 60% and reduce transform time by about the same proportion – 52%. It also reduces overall post-processing time, and memory requirement for storage. This is discussed further in *3.4 Effect of Thinning on Hough Transform Processing Time and Results*.

3.3 Artificial Neural Networks in Vision Systems for Mobile Robot Navigation

3.3.1 The Work of Chang and Others

The work (Chang et al 1994) discusses a system which enables navigation of a robot in unknown environments. Its navigation controller consists of three sub-controllers – the *main controller*, the *avoidance neural network* and the *forward neural network*. The controllers get input from infrared and ultrasonic sensors. The *main controller* checks whether or not the area ahead of the robot is safe and transfers control to the *forward neural network* or the avoidance neural network depending on what it finds.

The system is provided with an initial and a goal position and it works to get from one to the other while avoiding obstacles if necessary. (Chang et al 1994) conclude that neural network navigation controllers are efficient, robust and fault-tolerant.

The input to their system, infrared and ultrasound data, are different from the visual data employed in the current work. However, their use of feed forward neural networks and their conclusions about them are of interest to the current work.

3.3.2 The Work of Inigo and Others

The work of (Inigo et al 1995) is closely related to the current work in the sense that they use a single camera as input to a system they have developed. The system consists of three modules each performing one of three tasks - maintaining alignment, obstacle recognition and determining location of the robot relative to a fixed point of reference.

The system aims at robot navigation using “qualitative navigation behaviour”. The networks try to maintain the orientation of the robot while avoiding moving obstacles. The interaction of the three modules resulted in the robot moving in a zig-zag fashion.

A grading system they also developed compares the results of the modules with what a human referee decides is the correct response for the given situation. They report that the results for the system as a whole is better than the results for any of the modules individually.

Their use of neural networks to achieve navigation in a robot relates their work to the current one even though the way they use neural networks (for maintaining alignment, obstacle recognition and determining location) is different from the line detection approach of this work. The fact that they found that the overall result of the system is better than the result of the individual modules is similar to the situation with the system called hybrid 5 in this work discussed in *6.3.2 Hybrid 5: Use of ANNs to find lines in Sub-Images followed by use HT to establish ‘full-picture’*.

3.4 Works which Employ a Combination of the Hough Transform and Artificial Neural Networks

3.4.1 The Work of Dempsey and McVey

In an attempt to address the relatively slow processing speed associated with the (hardware) implementations of the Hough transform which has kept it from being widely used despite its importance as a robust and noise-resistant feature identification algorithm, (Dempsey and McVey 1992) have proposed the use of ANNs-like circuitry to map from image space to parameter space and a modified Hopfield optimisation network to detect peaks in Hough transform parameter space (two important but time consuming steps in the Hough transform). They report tremendous improvement in processing time. The current project only considers software implementations of the Hough transform and ANNs.

3.4.2 The Work of Yun and Others

(Yun et all 1998) set out to solve the problem of localisation – knowing the location of a mobile robot in motion. Localisation involves (1) finding major features such as walls and (2) either matching them to features on a built-in map and then updating the robots position and orientation information based on this match, or, where there is no match, adding the feature to the map.

They apply the Hough transform on sonar data to identify major wall-like features in environment using two different approaches.

In the first approach which they call the classic Hough transform, they use these steps:

- (i) transform sonar data points in Cartesian coordinate into curves in parameter space.
- (ii) superimpose a two dimensional grid over parameter space
- (iii) construct an accumulator array with elements corresponding to cells in parameter space grids (bins)
- (iv) find curves crossing each bin
- (v) update bin
- (vi) find points associated with each bin

(vii) fit line to recovered points using least squares method

In step two, they choose a range of [0,180] for θ and set an upper bound $\rho_{\max}$ for ρ to the maximum of the distances from the point of reference to all N sonar points in the experiment.

In the second approach, they use these steps :

(i) transform sonar data points in Cartesian coordinate into curves in parameter space
(ii) for each curve in parameter space, solve for intersections with curves from four neighbouring sonar points
(iii) check to see if these intersections are within a small neighbourhood of one another
(iv) cluster intersection points which result with a winner take all neural network

They conclude that using the classic Hough transform, the choice of resolution (for the accumulator array) is important and some resolutions are clearly not acceptable because they produce false features. However, they did not find any conclusive trend that would suggest a scheme for pre-selection of an optimum resolution based on accuracy requirements. Using the second approach, they can always find the lines required within a fairly constant margin of error.

One difference between (Yun et al 1998) and the current project, is that while they employ the Hough algorithm to detect features using data collected from sonar sensors, the current project uses visual image data collected from a camera. Another major difference between the two projects is that (Yun et al 1998) were primarily concerned with the problem of localisation of a mobile robot. Therefore, the more important features for them are the ones likely to match features on the map of their robot. These were generally large permanent structures. This project on the other hand is primarily concerned with the problem of self-navigation. Features of primary concern for this project would therefore be features such as corridor edges and doors which indicate paths which an autonomous mobile robot could take.

Their use of clustering rather than setting up an accumulator array and possibly applying traditional peak detection methods is probably only possible with sparse-type data such as sonar. With edge points such as used in this work, which tend to be much more numerous, and more densely packed, direct clustering of the intersection points of curves of four neighbouring points is simply infeasible – parameter space points near peaks typically have several layers of curves overlapping.

3.4.4 The Work of Meng and Kak

(Meng and Kak 1993) describe a vision guided mobile robot navigation system called NEUR-NAV that is "human-like" in two senses: (1) It can make do with non-metric models of the environment, i.e. modelling by the order of appearance and adjacency relationships of various landmarks is sufficient, and a geometric model of the environment is not necessary. Components of a hallway are defined by their semantic and functional significance rather than by geometry. (2) It can respond to human type commands like "follow the corridor turn right at the second T junction".

The system consists of a non-metric model of a hallway and a path planner. It also has a rule-based supervisory controller which activates and de-activates an ensemble of neural networks trained to each perform one of three primitive tasks - interpret visual information, follow a hallway and detect landmarks.

The hallway is modelled as an attributed graph. The nodes of this graph represent landmarks such as corridors, junctions, dead-ends, and doors. A node is represented by a list of attribute names and pointers. The links of the graph are also attributed and contain information about the distance between the physical landmarks represented by the nodes at their ends. For example, a corridor node contains the attributes name, primary direction, and pointers to the nodes which represent the faces, junctions, etc. that define the corridor. Figure 3.1 illustrates an example of a corridor node definition.

name:	C2

primary direction:	North
left node:	door, d176
	door, d175
	door, d174
	door, d173
	door, d172
right node:	power_panel, p3
	alcove, a179
	power_panel, p2
	alcove, a180
	bulletin_board, b2
behind node:	junction, J1
beyond node:	junction, J2

Figure 3.3 Example of a Corridor Node in (Meng and Kak 1993)

The graph is stored as an adjacency matrix with columns and rows pointing to nodes and elements pointing to the attribute value for the link between the nodes indexed by its column and the row.

The *hallway follower* consists of three sub-modules, the *Corridor Follower*, the *Junction_Left Follower*, and the *Junction_Right Follower* each consisting of two neural networks. Using two networks enables the module to turn without ‘overturning’. The *Corridor Follower* neural network take as input, data from Hough space after first implementing the Hough transform on a reduced version of the image.

The *Landmark Detector* module consists of a single neural network capable of detecting both junctions and dead-ends, and making qualitative estimates of distance between the robot and the landmark.

The system can process a camera image producing a navigational output within 2 seconds on an MC68030 processor.

Their use of neural networks as well as human type commands closely relates their work to the current project.

3.5 Other Related Works

3.5.1 The Work of Li and Others

Previous work on vision/navigation systems for mobile robots is well exemplified by the work of (Li W. et al 1996). They have developed a method for vision based navigation using a THMR-III outdoor mobile robot. Their method enables the robot which consists of a van with two cameras, two computers and no human beings, to recognise and drive down roads using knowledge integrated into a fuzzy rule base for edge detection. Like the current work, (Li W. et al, 1996) use a combination of Image Processing (edge-detection) and Artificial Intelligence (AI) techniques (fuzzy logic) to achieve self-navigation in a mobile robot based on visual information. The current work, looks at a combination of the Hough transform which is an image processing technique (which takes edge-detection a little further, to line detection), and Artificial Neural Networks (ANNs) which is an AI technique to achieve the same primary objective.

Chapter 4 Pre-processing

Various processes are required to get an image to the point where it is ready for application of the Hough transform, or an Artificial Neural Network. In image processing, it is common to refer to performance of these tasks as pre-processing. In this work, images are pre-processed to the point of obtaining a thinned edge-image for both a Hough transform vision system and an Artificial Neural Network vision system. This is because this work aims at developing vision systems which work by detecting and interpreting lines.

Pre-processing to the point of having a binary edge image is necessary for application of the Hough transform but not absolutely necessary for a neural network system. (Inigo et al 1995) for example, have developed a neural network based vision/navigation system for a mobile robot using grey-level images. The approach of (Meng and Kak 1993) employs lines, and in that regard is similar to the current work.

The pre-processing tasks used include resizing of the captured image, edge-detection and edge-thinning. These are discussed in the sections that follow. Image capture is also discussed here although it is not part of pre-processing in the classic sense.

4.1 Image Capture, Resizing and Conversion to Gray Scale

4.1.1 Capture-Process-Navigate Cycle

To achieve vision-based navigation, it is necessary to capture, and process an image, and then effect navigation on the basis of the result of the processing. This cycle is repeated until a predefined navigation programme is completely executed, or the entire navigation process is terminated.

4.1.2 Capture

In the current work, images were captured using a single forward-facing camera.
It was ensured that there was sufficient light to clearly identify separate features in

the images such as walls, floors and doors. The base of the camera was set up parallel to the floor.

4.1.3 Resizing

A standard image size was chosen to give a good compromise between usefulness of output and processing time. The reduced image size chosen was 128 x 96 pixels. When this size of image is fully processed, fairly fine details such as the two edges of a door on the side of a corridor can be extracted, yet the time for processing the image is not prohibitive.

Other image sizes were tried. These included 32x32 pixels and 64x48 pixels. In both cases, the level of detail available when the image is fully processed is limited and means that high level post-processes do not have adequate input. A feature such as a door that is noticeable to a human observer in an image can be reduced to a single line if the image is reduced to a 32 x 32 size, and so the door cannot be picked up as a door by the post-process for detecting doors, for example. Figure 4.1 shows the various types of results for a typical image. Figure 4.1a is the original image magnified by 2.67, figure 4.1b is the 32 x 32 thinned version magnified by 8, figure 4.1c is the 64 x 64 version magnified by 4 and figure 4.1d is the 128 x 96 version again magnified by 2.67. The door circled in figure 4.1a has no chance of being picked up as a door in the 32 x 32 thinned image because it almost doesn’t appear, and in the 64 x 64 thinned image because it appears as a single line.

Also, although a square aspect ratio was considered, a 4:3 aspect ratio was selected as the cameras used all captured in 4:3 ratio and changing the ratio led to unnecessary loss of information from the sides as can be seen by studying figure 4.1.

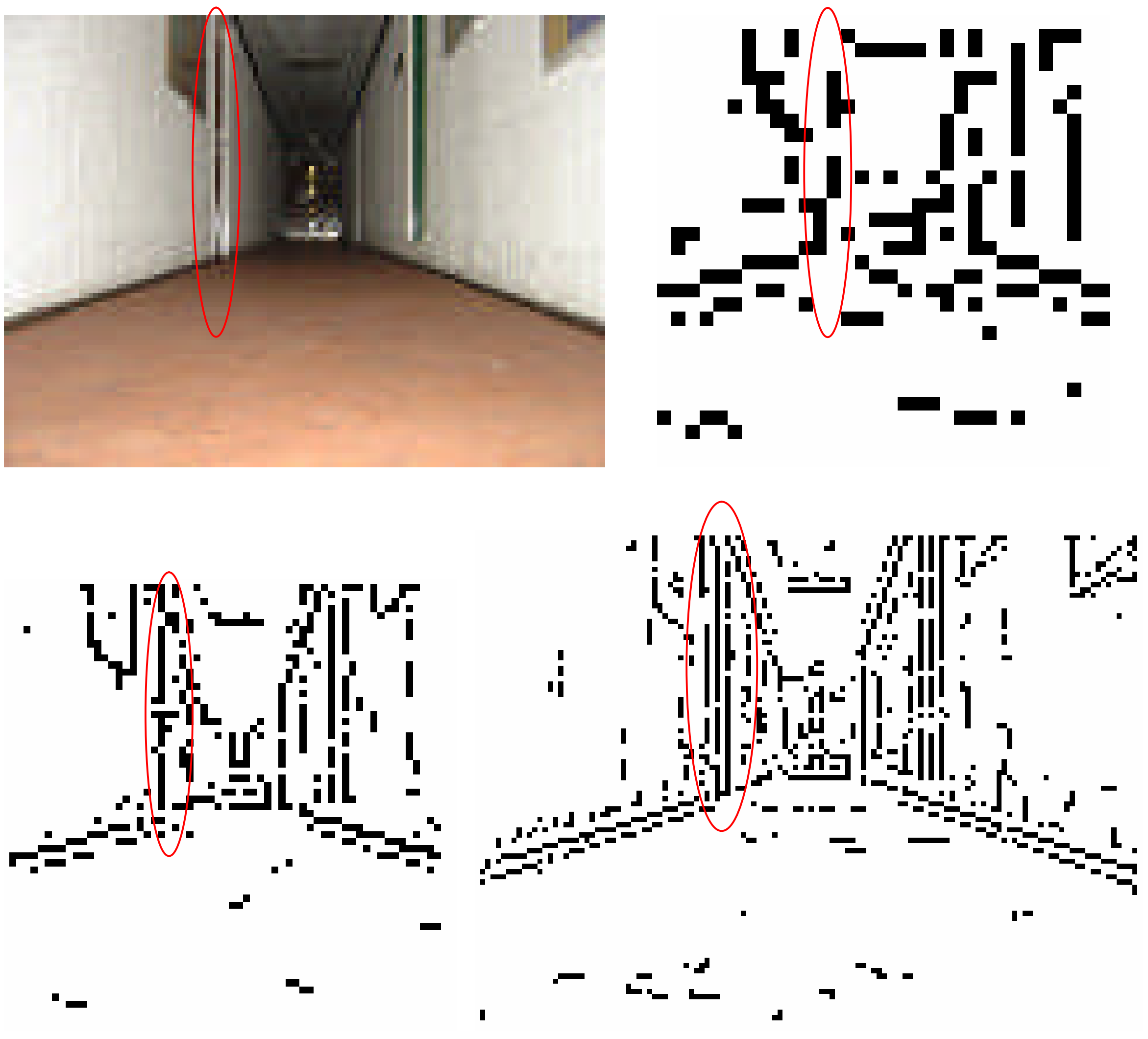


Figure 4.1 Effects of various image sizes
(a) Original Image magnified by 2.67 (b) 32 x 32 thinned image magnified 8 times (c) 64 x 64 thinned image magnified 4 times (d) 128 x 96 thinned image magnified by 2.67

Lower resolutions than used in this work have been used by other works. (Inigo et al 1995) used a 30 x 32 sized grey-scale image as input to a neural network for the purpose of navigating a robot to avoid moving obstacles and turning into junctions. (Meng 1993) reduce their 512 x 480 sized images to 64 x 60 sized images in their *Corridor Follower* module and then use them as input for the Hough transform. They then use the resulting Hough space as input to a neural network. They report that the reduction in size has no noticeable effects on the performance of the module.

(Vaughn and Ronald 1990) resized their capture 512 x 512 sized image to a 256 x 256 size – a higher resolution than the images in the current work - and use it to locate a docking station using an algorithm that requires up to 5 runs of the Hough transform. They report very high processing times (up to 10 minutes) however.

4.1.4 Intensity Determination

The camera used for this work captures coloured images. This are stored as image objects which have information about the levels of red, blue and green at every point of the image. For edge-detection to commence, it is necessary to determine the intensity at each point. This is done by extracting the level of each of the three colours and determining the average at each point.

4.1.5 Image Point Indexing

Points in images are labelled with identification codes illustrated in figure 4.2. The point at the top-left position is labelled 0. Subsequent points going right are labelled with consecutive numbers until the end of the row. The labelling is continued on the next row from the left.

0	1	2				125	126	127
128	129	130	.	.	.	253	254	255
256	257	258				381	382	383
	.						.	
	.						.	
	.						.	
11904	11905	11906				12029	12030	12031
12032	12033	12034	.	.	.	12157	12158	12159
12160	12161	12162				12285	12286	12287

Figure 4.2 Image points indexing

4.2 Edge Detection

Edge-detection is the first pre-processing step implemented after an image of the right size has been obtained. It yields an edge-image by plotting lines connecting points where there are significant changes in pixel intensity, and which can therefore be taken as reliable indications of edges of features in the image (Leavers 1992). An edge image, ideally, contains lines that outline features in the original image.

With the intensities in the grey-scale image determined as discussed in *4.1.4 Intensity Determination*, a filter is applied across the image, which measures for each point in the image, the possibility that the point is an edge. A threshold, selection of which is a task in itself, is then applied to select points with high possibilities of being edge points. *1.1.2 Pre-Processing* briefly introduces edge-detection in image processing.

As discussed in *1.1.2 Pre-processing*, the Sobel edge-detection filters were chosen for this work. *1.1.2 Pre-processing* introduced the filters, and this section presents further details about its use in this work. A fuller discussion on the Sobel filters is available in (Leavers 1992), as well as several other resources.

The Sobel filters are two 3 x 3 matices, M_{ver} and M_{hor} and these are applied across the image.

M_{ver} is defined as

$$M_{ver} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad . \quad . \quad . \quad \text{Equation 4.1}$$

and is designed to find vertical edges and M_{hor} defined as,

$$M_{hor} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad . \quad . \quad . \quad \text{Equation 4.2}$$

is designed to find horizontal edges.

The filters yield a measure of the possibility that there is a vertical and a horizontal edge, respectively, at a given point. These measures are called gradient magnitudes. The two gradient magnitudes, gm_{ver} and gm_{hor}, are obtained by convolution of the respective filters with the image I:

$$gm_{ver} = \left| M_{ver} * I \right| \qquad . \qquad . \qquad . \qquad \text{Equation 4.3}$$

$$gm_{hor} = \left| M_{hor} * I \right| \qquad . \qquad . \qquad . \qquad \text{Equation 4.4}$$

The two are then summed to give an overall gradient magnitude, GM for the point:

$$gm = gm_{ver} + gm_{hor} \qquad . \qquad . \qquad . \qquad \text{Equation 4.5}$$

The Sobel filters also provide and an estimate of the angle, α of the gradient. This is simply the arc tangent of the horizontal gradient magnitude divided by the vertical gradient magnitude:

$$\alpha = \tan^{-1}\left(\frac{gm_{hor}}{gm_{ver}} \right) \qquad . \qquad . \qquad . \qquad \text{Equation 4.6}$$

4.2.1 Edge Threshold determination

Once gradient magnitudes have been determined, the next stage in edge-detection is deciding from the gradient magnitudes, which points are edge points and which ones are not. This involves application of a threshold. Rather than assign a fixed threshold for determining edges, a target is provided of the number of edge points required. The following sequence of steps is then used to work out what threshold will result in getting a number of edges close to that specified:

1. Determine maximum gradient magnitude, M, from the array of gradient magnitudes GM
2. Determine minimum gradient magnitude, m, from the array of gradient magnitudes GM
3. Determine range of gradient magnitudes, R, using $R = M - m + 1$
4. Determine target number of non-edge points, N', as the difference between total number of points, N, and target number of peaks, T', i.e. $N' = N - T'$
5. Determine the number of elements of GM having value a where $m \leq a \leq M$ and store as G_a

6. Initialise a counting variable i to m, and set S_i, the m^{th} cumulative sum, to G_m
7. Increase i by 1
8. Add previous cumulative group count to current group count to get current cumulative group count, i.e., $S_i = S_{i-1} + G_i$
9. If current cumulative group sum, S_i, is equal to target number of non-peaks, N', do 10a, if it is greater do 10b, else go back to 7

10a. Set threshold to the last valid count

10b. Set threshold to the current count

The gradient magnitudes determined by application of the Sobel edge-detection filters, provides input for this algorithm.

A cumulative sum of groups is taken until a sum is reached which equals or just exceeds (number of bits – target number of edges). The average of the last non-zero gradient magnitude is then taken as the required threshold. This has the effect of returning a threshold that will return a number of edge points equal to, or the nearest above the targeted number of edge points.

Sample results are shown in figure 4.3. Figure 4.3a is a typical image, and figure 4.3b is the same image after it has been converted to grey-scale and Sobel edge detection has been applied to it.

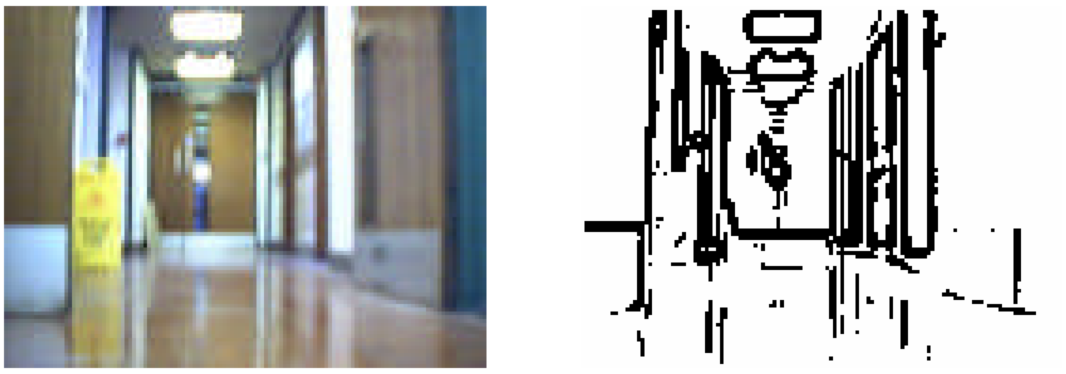


Figure 4.3 Sample Sobel edge detection results
(a) Sample Image (b) Sample Image after Sobel Edge Detection

4.3 Edge Thinning

Edge-detection often yeilds edges several pixels thick. This can make further processing of the image unnecessarily processing time and memory consuming, and “distracts” feature detection processes from important but salient features of the image. The objective of edge thinning is to reduce edges to unit thickness without losing any information about the connectedness of edges or introducing any form of distortion to the image.

Several thinning algorithms exist. The most popular method is the non-maximum suppression method. This method works by removing edge responses that are not maximal in each cross section of the edge direction in their local neighbourhood. However, the result of this method is still under-thinned in some places and removes real edges in other places (Park and Chen 2000).

(Park and Chen 2000) have proposed another method based on comparing gradient magnitudes within 3 x 3 neighbourhoods. It produces more accurate results than the non-maximum suppression method, and also has the added advantage of minimising the use of the edge direction, which introduces a lot of arc tangent calculations.

This work found that the method of (Park and Chen 2000) produces very good thin edges except that sometimes it loses information about edges that are significant in the context of the original image, and that would also be helpful for robot navigation. A slight modification has been proposed to step 1 of their method that has solved this problem.

Steps 0 and 1 of their method follows:

```
Step 0: Select an unprocessed edge point

Step 1: Determine number of edge points, $n$, in the immediate neighbourhood of the current point.
        If $n \leq 2$, set current point to a non-edge point, i.e., consider as noise
        else, go to step 2.
```

The modification to step 1 is:

```
Step 0: Select an unprocessed edge point

Step 1: Determine number of edge points, $n$, in the immediate neighbourhood of the current point.
    If $n = 0$, set current point to a non-edge point.
    If $n = 1$, then find out the number of neighbouring edge points, $nn$, of the 1 neighbour.
        If $nn > 1$, the current edge point is maintained otherwise it is made a non-edge point.
    If $n = 2$, maintain as edge point
    If $n > 2$, go to step 2.
```

Further processing is done exactly according to step 2 and further steps described in (Park and Chen 2000). Figure 4.4 show sample results. Figure 4.4a is a sample image. Its edge image is shown in figure 4.4b. The results of the algorithm of (Park and Chen 2000) are shown in 4.4c and the results of the modification by the current work. Although the method of (Park and Chen 2000) results in a cleaner result, it looses important lines such as the door border highlighted in figure 4.4b.

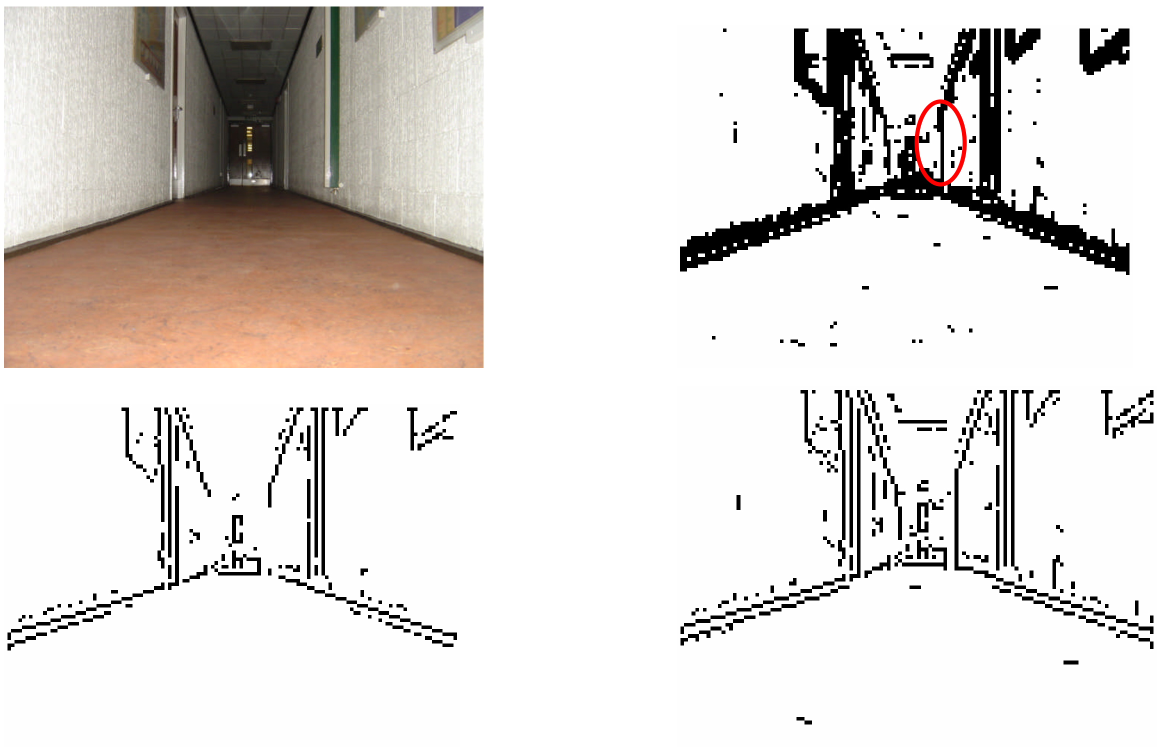


Figure 4.4 Comparison of the results of the thinning method of (Park, 2000) and the modified version of it used in this work
(a) Sample capture image (b) Sample image after application of the Sobel Operator (c) Sample image thinned with the method of (Park, 2000) (d) Sample image thinned with modification to the method of (Park, 2000) by this work

4.4 Effect of Thinning on Hough Transform Processing Time and Results

A simple experiment was done to illustrate the effect of thinning on the processing time and quality of results of subsequent processes. A typical image was pre-processed to two points - to the point of edge detection, and to the point of thinning. The version with pre-processing done to the point of edge detection was labelled NT (for no thinning) and the version pre-processed to the point of thinning was labelled WT (for with thinning).

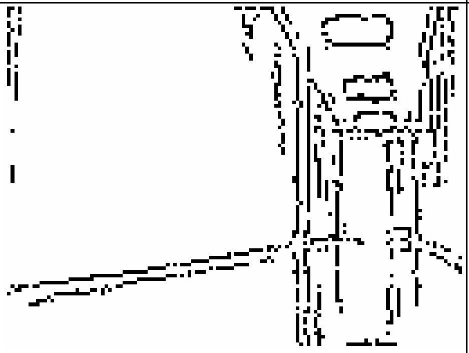
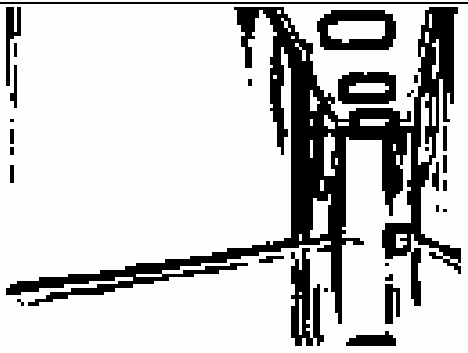
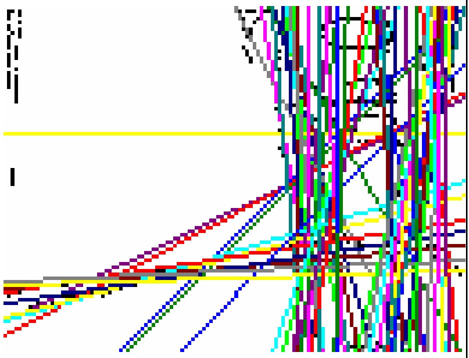
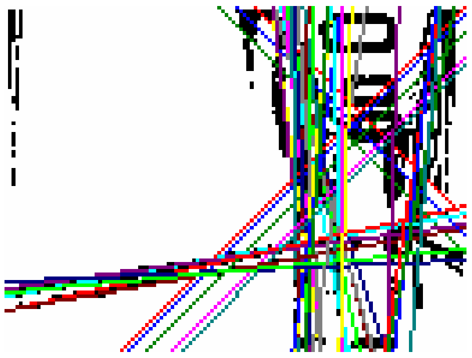
Both NT and WT were then further processed with the Hough transform, and post processed to the point of determining valid sub images. Both runs were done twice so that variations in timing can also be monitored. Results for the first and second runs for NT are labelled NT1 and NT2 respectively. The same labelling scheme was used for WT. Outcomes of these are discussed in *4.4.1 Effects of Thinning on Quality of Results of Subsequent Processes* and *4.4.2 Effects of Thinning on Processing Times of Subsequent Processes*.

The processes involved are discussed in Chapter 4, but they are used here to illustrate the effect thinning or lack of it can have on further processes.

4.4.1 Effects of Thinning on Quality of Results of Subsequent Processes

Table 4.1 summarises results from the experiment to illustrate the effects of thinning. Row 1 of the results shows the take off number of pixels after Sobel edge detection, 2491. The next row shows the number of pixels after thinning. This is only applicable to WT as NT was not thinned. Row 3 following that illustrates the two versions of binary images that were used as input to the Hough transform. Lines in NT are noticeably thicker than lines in WT.

Row 4 states the number of peaks found when the peak detection scheme used in this work was applied with a target number of peaks of 200. 236 peaks resulted from WT and 206 from NT. Row 5 shows that 38 and 22 lines respectively resulted when the butterfly filter is applied and local maxima are selected in 5 x 5 neighbourhoods in the way done by this work as discussed in Chapter 5.

Process		Pre-processing type	
		With thinning (WT)	**No thinning (NT)**
1	Number of edge pixels	2491	2491
2	Number of edge pixels after thinning	1008	-
3	Input Image for Hough transform		
4	Number of peaks found	239	206
5	Number of local maxima within 5x5 neighbourhood	38	22
6	Lines corresponding to local maxima in 5x5 neighbourhood superimposed on HT input image		
7	Number of sub-lines found	40	57
8	Number of critical sub-lines found	6	5

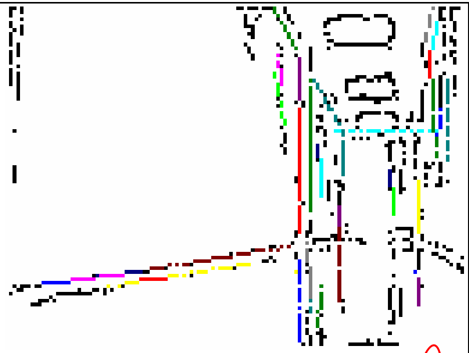
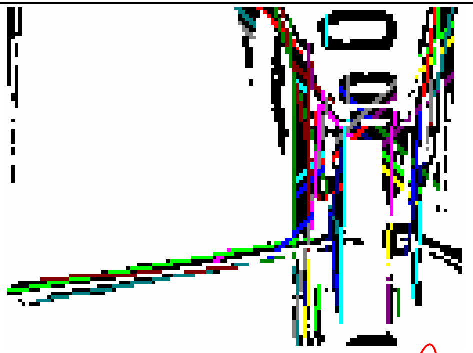
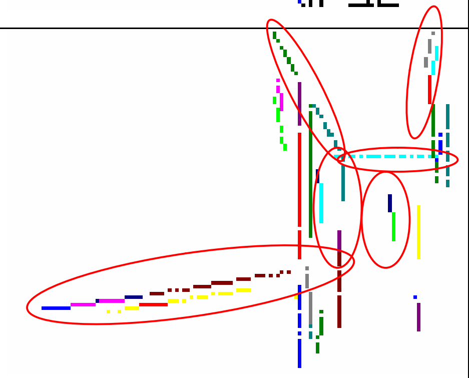
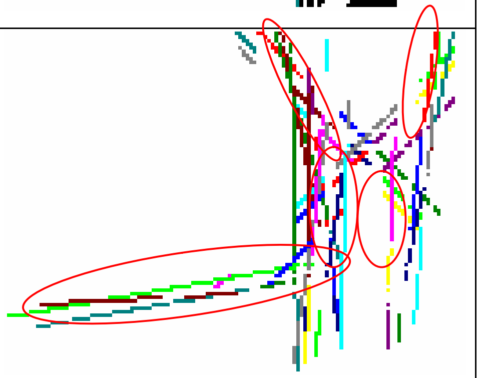
9	Sub-lines found superimposed on HT input image		
10			

Table 4.1 Results of processing to the point of sub-lines detection from thinned and "unthinned" edge image

A few interesting observations can be made. Firstly, as stated in row 2, thinning results in a significant reduction of edge points from 2491 to 1008. This represents approximately, a 59.53% reduction. This, as is seen in *4.4.2 Effects of Thinning on Processing Times of Subsequent Processes* which follows, has a significant effect on Hough transform time, and ultimately on the time taken for the entire process up to sub-line detection.

Secondly, sub-lines found in WT appear to match lines in the image that actually represent high level features, to a higher extent than sub-lines in NT. In particular, more navigation critical sub-lines were found in WT than NT as state in row 8 and illustrated in rows 9 and 10 even though more sub-lines were found in NT as stated in row 7. Navigation critical sub-lines are those that will actually aid detection of doors, and corridors possible for the robot as discussed in *5.3 High Level Features Determination*, and therefore possibly affect its navigation decisions as discussed in *5.4 Navigation*. They are illustrated in figure 4.5.

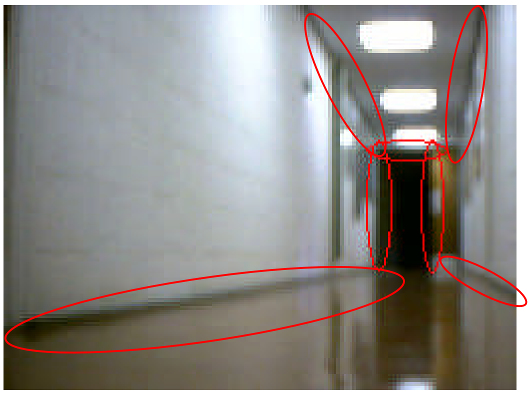


Figure 4.5 Navigation critical lines in a typical image

4.4.2 Effects of Thinning on Processing Times of Subsequent Processes

Tables 4.2 shows the timings recorded for a number of subsequent processes. An Intel Duo T5600 1.83GHz processor machine with 1GB of RAM was used for this experiment.

Process Times (milliseconds)		**Pre-processing type**							
		WT 1	**WT 2**	**Difference between WT1 and WT2**	**Average of WT1 and WT2**	**NT 1**	**NT 2**	**Difference between NT1 and NT2**	**Average of NT1 and NT2**
1	Conversion to grey scale time	1100	1090	10	1095	1090	1100	10	1095
2	Edge detection time	1090	1090	0	1090	1080	1090	10	1085
3	Thinning time	0	160	160	80	-	-	-	-
4	HT time	1090	1090	0	1090	2190	2340	150	2265
5	Peak detection time	160	310	150	235	310	320	10	315
6	Butterfly filter application time	150	160	10	155	160	160	0	160
7	Time to find	0	160	160	80	150	0	150	75

	sub-lines								
8	Total time taken	3590	4060	470	3825	4980	5010	30	4995

Table 4.2 Comparison of process times for an image when it is thinned and when it is not

Three issues were looked at in terms of processing time - (1) what is the cost of thinning, (2) what benefits can be derived from thinning, and (3) what is the overall effect of thinning. These are discussed in *4.4.2.2 Costs of Thinning*, *4.4.2.3 Benefits of Thinning* and *4.4.2.4 Sum Effect of Thinning on Time Taken* respectively. Before going there however, a look is taken at errors in the measurement of time in *4.4.2.1 Errors in Time Measurement* below.

4.4.2.1 Errors in Time Measurement

Before discussing the results in relation to the purpose of the experiment, two unexpected observations need to be made regarding the accuracy of the time recording mechanism used to set up table 4.2. First for some processes 0 milliseconds was recorded as time taken. These include thinning time and time taken to find sub-lines for WT 1, and time taken to find sub-lines for NT 2. This suggests that the time recording mechanism is limited in its sensitivity. The extent of this insensitivity is not certain but is probably hinted by other times recorded for exactly the same events. Thinning time for WT 2 was 160 milliseconds, time taken for determining sub-lines for WT 2 was 160 milliseconds, and time taken to determine sub-lines for NT 1 was 150 milliseconds. These suggests that events that have up to 160 milliseconds as time recorded for them to complete may at other times have as low as 0 milliseconds recorded as the time to complete them.

The second related unexpected issue is that exactly the same processes with the same data take different amounts of time to complete at different times. The first two rows of results in table 4.2 show the time taken to convert the image to grey scale and the time taken to perform Sobel edge detection. The same amount of time was expected for all four runs of the two processes. However, there were variations of up to 10 milliseconds. Further down the table, other unexpected variations were also observed. The time recorded for peak detection for WT 1 was 160 and for WT 2 it was 310 milliseconds meaning there was a 150 milliseconds difference. Other unexpected variations were those between butterfly filter application times for WT 1 and WT 2 (10 milliseconds), between Hough transform times for NT 1 and NT 2 (150 milliseconds), and between peak detection times for NT 1 and NT 2 (10 milliseconds). These are in addition to the variations involving 0 times already mentioned. These differences are likely to be due to differences in work load on the processor due to background processes at the times that the process times being studied were recorded.

These two observations suggest that errors of up to 160 milliseconds can be expected, and so differences in time of up to that figure should not be taken as significant.

4.4.2.2 Costs of Thinning

The most obvious cost of thinning would be the time taken for thinning in WT. From row 3 of table 4.2, the average time taken for thinning is 80 milliseconds although it can be up to 160 milliseconds. Factoring in the error margin adapted in *4.4.2.1 Errors in Time Measurement* above, this time can be ruled as insignificant.

Another way to see it is: Time taken for edge detection WT on the average is 1090 milliseconds. Time taken for thinning is 160 milliseconds. Therefore thinning increases binary image derivation by a factor of 160/1090 which is approximately 14.68%.

Other than thinning itself, there are only two other processes for which WT records higher time than NT. One is edge detection (row 2) where there is a difference of 5 milliseconds. This is insignificant as well as irrelevant because edge detection happens before thinning. The other is the time taken to find sub-lines (row 7), again higher by an insignificant 5 milliseconds. Although the difference is insignificant, it is not unexpected that WT records higher time here than NT as the number of input lines for that process 38, is higher than 22 for NT as row 5 of table 4.1 shows.

4.4.2.2 Benefits of Thinning

In table 4.2, the most significant saving of time due to thinning happened with the application of the Hough transform where NT required on the average 1175 milliseconds more than WT. Put another way, a saving of 1175 milliseconds represents a saving of about 52% of the 2252 milliseconds taken by the Hough transform for NT on the average. This is not unexpected as the number of times the core of the Hough transform algorithm runs is directly proportional to the number of edge pixels in its input image. WT had 2495 – 1008, i.e. 1487 fewer edge points than NT (row 3 of table 4.1). This represents a 60% reduction and correlates quite well with the 52% savings on time.

The process whose average NT time is higher with the next highest value is peak detection. It is higher by 80 milliseconds which by the error margin established in 4.4.2.1 is not significant.

Other processes for the thinned WT recorded lower times include conversion to grey scale – a 5 millisecond difference that is irrelevant as the process happens before thinning - and butterfly filter application also with an insignificant 5 millisecond difference.

4.4.2.2 Sum Effect of Thinning on Time Taken

Thinning had no significant effect on time taken for individual processes except for the Hough transform. This effect is so significant it reflects in the average total time row of table 4.2. There is a difference of 1170 milliseconds which is very similar to the 1175 millisecond difference of the Hough transform. 1170 milliseconds represents (1170/3825)%, i.e, 30.59% of the total time taken by WT.

Thinning makes a significant difference to the amount of time taken.

Chapter 5 A Hough Transform Vision System for a Mobile Robot

Chapter 5 discusses a Hough transform based vision and self navigation system for a mobile robot. *5.1 Implementation of the Hough Transform* discusses the implementation of the Hough transform as it is done specifically in this work. Post processing and labelling of lines is then discussed in *5.2 Post Processing*, and detection of high level features such as corridors and doors is discussed in *5.3 High Level Features Determination*. Section *5.4 Navigation* then discusses navigation issues.

5.1 Implementation of the Hough Transform

In this section, specific implementation of the Hough transform for this project is discussed.

As discussed in Chapter 1, a key element of the Hough transform is setting up the accumulator array. The accumulator array is a table of θ values, and corresponding ρ values. To set it up, first a decision is made about the angular resolution to be used. This is discussed in *5.1.2 θ Resolution*. Secondly, to be able to model the array in a computer programme, the range of corresponding ρ values is determined. This is discussed in *5.1.3 Range of the Hough Transform Function*.

With the accumulator array in place, the Hough transform can be applied to the thinned image derived from the pre-processing described in Chapter 4. This assigns different accumulations to elements of the accumulator array, and is discussed in *5.1.5 Transformation and Accumulation*. To achieve the goal of finding important lines from the original image, peaks have to be detected in the accumulator array, and reverse transformed. Peaks are determined by application of a threshold which is itself worked out for every image to get best results. Peak detection is discussed in *5.1.6 Peak Detection*.

5.1.1 Coordinates in Image Space

In applying the Hough transform, it is usual for coordinates for points in image space to be chosen so that the origin (0, 0) is at the centre of the image. That approach is adopted in this work. Given, from the discussion in *4.1.3 Resizing*, that the width of the image is 128 pixels and the height is 96 pixels, it follows that x ranges from -64 to 63 increasing from left to right, and y ranges from -48 to 47 increasing from bottom to top. From the indexing scheme discussed in *4.1.5 Image Point Indexing*, if follows that for a point with index i,

$$x = iMod128 - 64 \qquad . \qquad . \qquad . \qquad \text{Equation 5.1a}$$

and

$$y = 47 - i/128 \qquad . \qquad . \qquad . \qquad \text{Equation 5.1b}$$

where the operators Mod and $/$ refer to the integer remainder and integer division operations respectively. The first (top left) point in the image with i=0, has coordinates (-64, 47), for example, and the origin (0, 0) is the point with index i= 47*128 + 64 i.e. i=6080.

5.1.2 θ Resolution

An early step in the application of the Hough transform, as discussed in *1.1.3 Transformation from Image Space to Hough Space*, is deciding what step value to use for θ. The aim is to choose a resolution that will make it possible to find any reasonably significant line possible in image space. This means that a point A at the centre of the image which is taken to be the origin, should be seen to be collinear with another point B at the far end of the image, and also be seen to be collinear with another point C right next to point B. Figure 5.1 is an extract from an illustration image with the regular size magnified 8 times to show three such points. The green square A is a pixel at the centre of the image, the red point B is a pixel at the farthest possible distance from A and the blue one C is the pixel next to the red one.

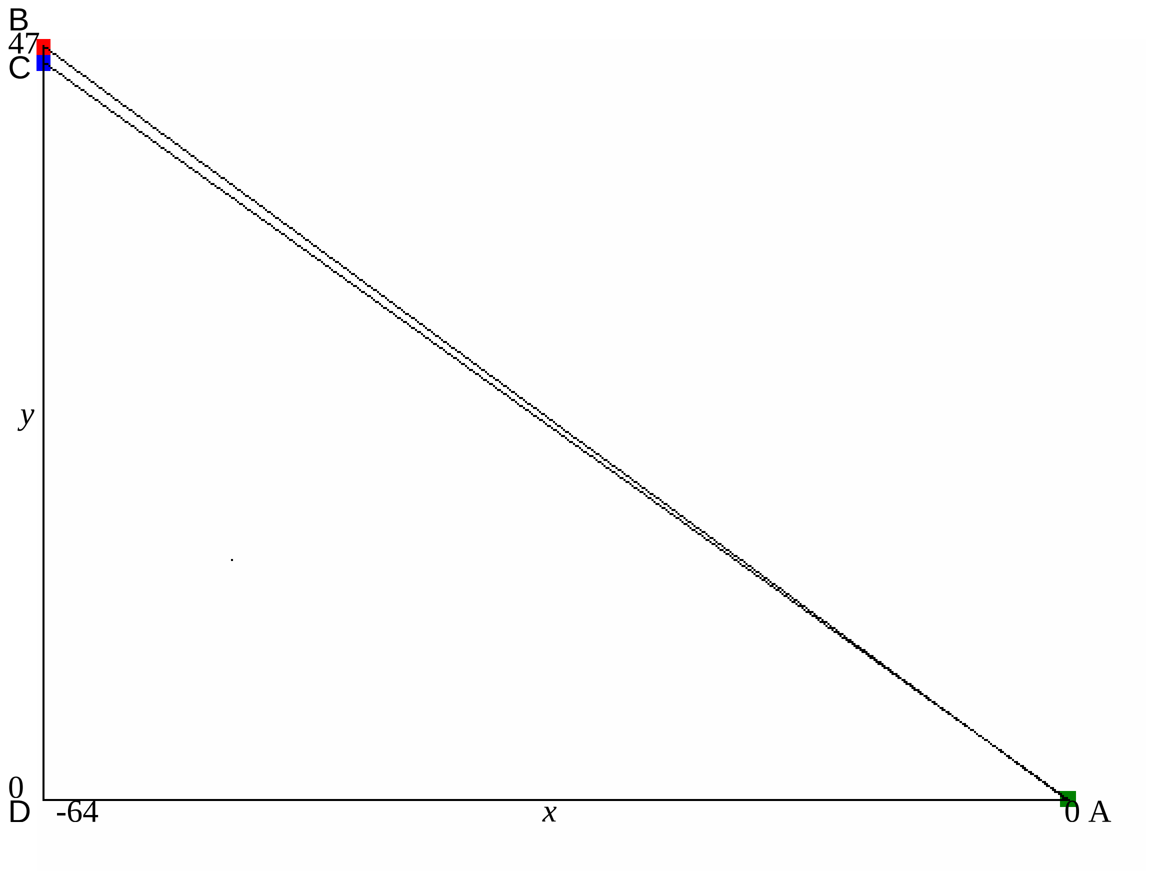


Figure 5.1 Angle due to 1 pixel difference at farthest distance from origin

The minimum change in angle $\Delta\theta$ corresponding to the angle BAC, needs to be so small that the line AB or any sub-line on it can be picked up, as well as the line AC or any of its sub-lines. From the dimensions of the image, AD is 64 and BD is 48. Therefore the angle BAD will be $\tan^{-1}(47/64)$, i.e., 36.2926°. Similarly, angle CAD will be $\tan^{-1}(46/64)$, i.e., 35.7067°. $\Delta\theta$ which is the difference of the two angles is 0.5859°. This is the value of $\Delta\theta$ that can pick up both lines AB and AC. However, for the purpose of setting up indices for an accumulator array, an integral value is required, so 1° is the required interval, being 0.5859°rounded to the nearest whole number.

Figure 5.2 shows lines plotted centred at the origin, at intervals of 1°. The full image is covered – very densely towards the middle where the multiple-peak effect described by (Leavers 1995) can be expected, and a little sparsely towards the edge. This can be expected because the actual interval of 1° is higher than the ideal interval of 0.5859°. It can be observed though that there is no uncovered part of the image wider than 1 pixel. Points that should have fallen in those spaces would have been quantised to adjacent pixels. For the purpose of comparison with figure 5.2, figure 5.3 was also developed with a 2° interval. Towards the diagonals, spaces of up to 3 pixels in width exist, and lines of length up to 8 pixels can be missed if that interval is used.

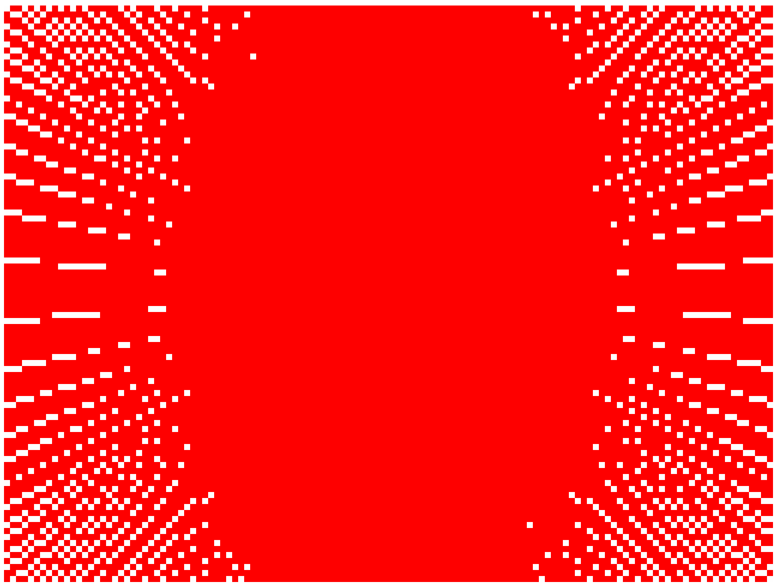


Figure 5.2 Lines drawn across the range of θ at 1° interval

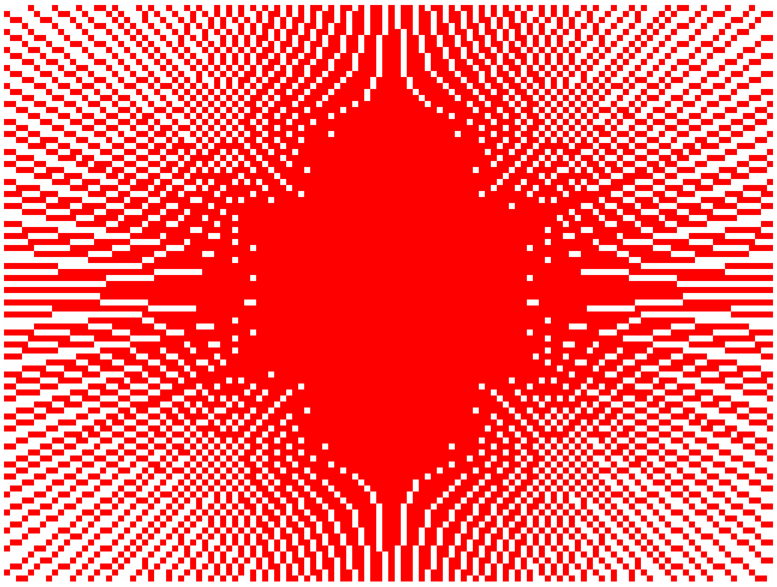


Figure 5.3 Lines drawn across the range of θ at 2° interval

5.1.3 Range of the Hough Transform Function

As pointed out in *1.1.3.2 Straight Line Hough Transform Using the Normal Form of the Equation of a Straight Line*, values of the parameters ρ and θ are bounded unlike m and c when equation 1.1 is used as the basis for transformation. These bounds dictate the size of the parameter space and the accumulator array. It was further pointed out that θ lies in the interval $[0°,180°)$ and ρ lies in the interval $[-\sqrt{(h^2+w^2)},\sqrt{(h^2+w^2)})$ where h and w are as defined in *1.1.3.2 Straight Line Hough Transform Using the Normal Form of the Equation of a Straight Line*. For the 128 x 96 pixel image being used, h is 48 and w is 64. ρ is therefore bounded by $\left[-\sqrt{(64^2+48^2)},\sqrt{(64^2+48^2)}\right)$ or $[-80,80)$.

5.1.4 Accumulator Array

With the bounds of θ and ρ known, the accumulator array is set up. θ ranges from 0 to 180°s as discussed in 5.1.3 above. However, as discussed in *5.1.6 Peak detection*, it is helpful to take a range for θ up to 185°. This makes peak detection techniques such as application of the butterfly filter easier to apply.

As discussed in *5.1.2 θ Resolution*, θ is increased at an interval of 1°, and ρ values are quantised to a single pixel as discussed in *1.1.3.4 Parameter Quantisation Intervals*. Also from the discussion in *5.1.3 Range of the Hough Transform Function*, the range of values for ρ is $[-80,80)$. A resolution of 1 is selected for maximum accuracy. With these considerations in mind, the accumulator array can be set up as shown in figure 5.4 indicating the ranges of θ and ρ.

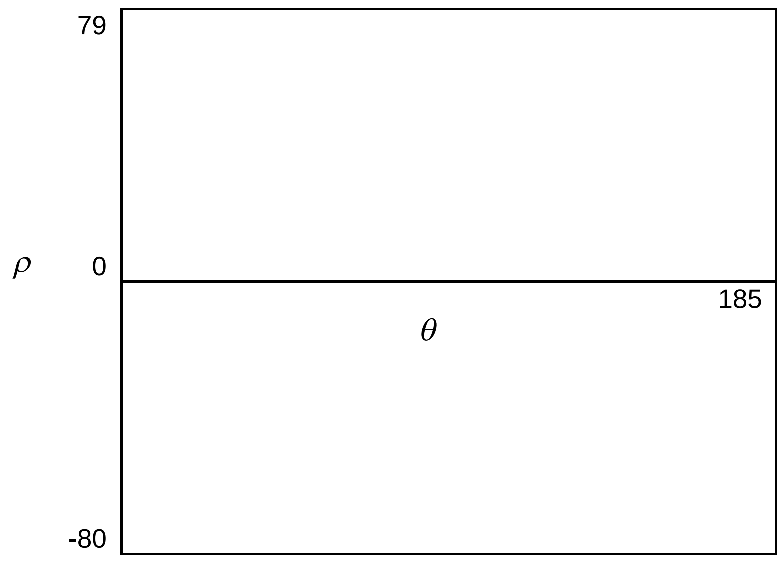


Figure 5.4 Ranges of θ and ρ in Accumulator Array

5.1.5 Transformation and Accumulation

With the accumulator array in place, the transform proper can commence. The following sequence of steps describes this implementation.

1. Initialise accumulator array entries to 0
2. For all points of image space, do 3
3. If point is black do 4 to 5 else do 6
4. For θ from 0 to r, the upper boundary of θ, do 5 and 6
5. Determine ρ using $\rho = x\cos\theta + y\sin\theta$
6. Increase the accumulator array entry for current θ and ρ

5.1.6 Peak Detection

As pointed out by (Grimson 1990), the Hough transform can yield false peaks, sometimes in significant proportions.
Peak detection is achieved by a scheme consisting of a number of stages:

1. Determination and application of the most appropriate threshold for the image under consideration
2. Application of the butterfly filter to entries above the threshold from step 1 above only
3. Determining which elements of the accumulator array selected from 2 above are local maxima within a 5 x 5 neighbourhood

This scheme is slightly different from any other schemes encountered in the literature, and combines the real-time variable threshold determination process described described shortly which has also not been encountered in any previous literature, with straightforward threshold application traditionally used with the Hough transform, and also with the butterfly filter approach of (Boyce et al 1987).

The scheme is necessary because, as pointed out by (Grimson 1990) and (Leavers 1995), the Hough transform can yield false peaks in high proportions and multiple peaks respectively, and is one of several measures taken by this work to minimise the emergence or the effects of false and multiple peaks.

These stages of the scheme are discussed in detail in *5.1.6.2 Threshold Determination*, *5.1.6.3 Application of the Butterfly Filter* and *5.1.6.4 Determination of local maxima*. Before them, *5.1.6.1 Determination of Target Number of Peaks* discusses the choice of the target number of peaks which is used in determining threshold in *5.1.6.2*.

5.1.6.1 Determination of Target Number of Peaks

This work uses a simple algorithm to work out the most appropriate threshold to employ given a target number of peaks. A study was done and results were recorded for three typical images labelled image1, image2 and image3, to determine a reasonably good target number of peaks for the system. A good target would eventually lead to the detection of navigation critical lines while keeping processing time and memory use as low as possible.

Each image was processed with a target number of peaks of 100, 150, 200, 250 and 300. A discussion of results for each image follows.

Image1 is displayed in figure 5.5a and the pre-processed version of it is shown in figure 5.5b. In both figures, navigation critical lines are circled in red. Figure 5.6 shows results with target number of peaks set at 100. Figure 5.6a shows the lines found from peak detection and subsequent application of the butterfly filter, and figure 5.6b shows sub-lines found in colours other than black. Similarly, figures 5.7, 5.8, 5.9 and 5.10 show the results for the cases with target number of peaks set at 150, 200, 250 and 300 respectively. Table 5.1 shows further details for each of the images.

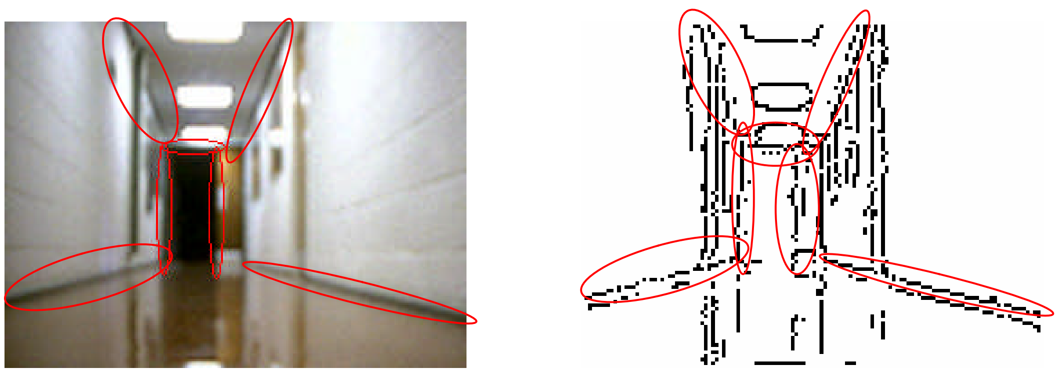


Figure 5.5 Navigation critical lines in a typical image – image1
(a) A typical image (b) Thinned version of image1

Figures 5.6a and 5.6b show that with a target number of peaks of 100, 6 out of 7 navigation-critical sub-lines were found. The line corresponding to the top of the circles in blue was not detected (it is still completely depicted by black pixels).

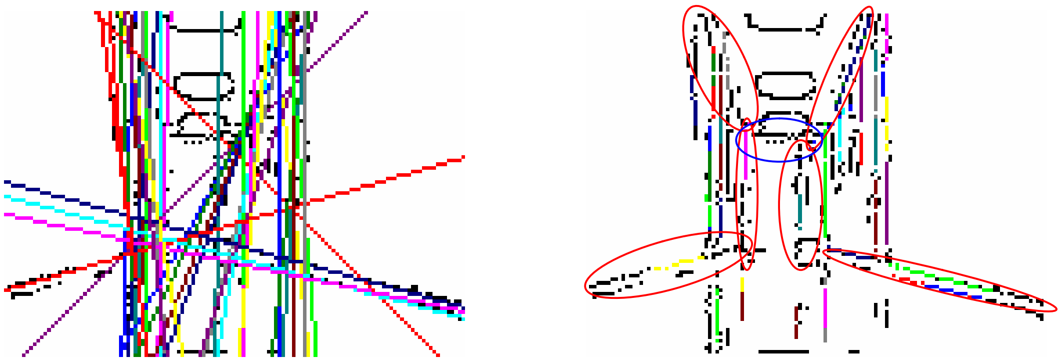
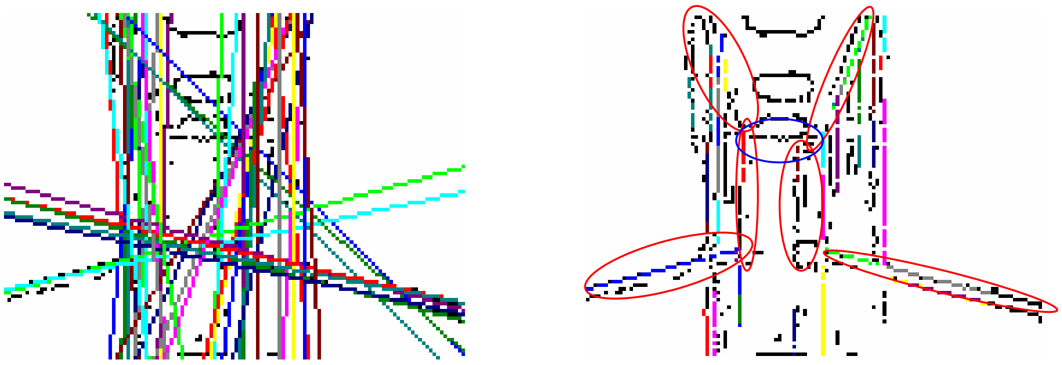


Figure 5.6 Lines and sub-lines found in image1 with T=100
(a) Lines (b) Sub-lines

The same number of navigation-critical lines was found when T was increased to 150 as illustrated in figures 5.7a and 5.7b.



F**igure 5.**7 Lines and sub-lines found in image1 with T=150
(a) Lines (b) Sub-lines

When T was set at 200, 250 and 300, all 7 out of 7 navigation critical lines were found as shown in figures 5.8, 5.9 and 5.10.

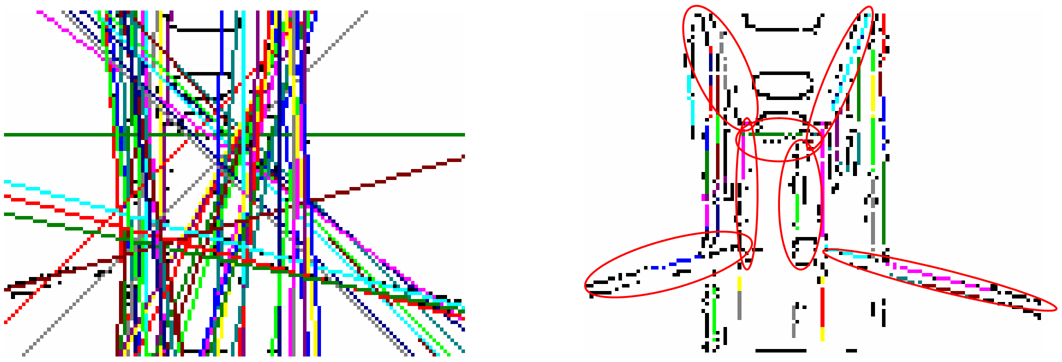


Figure 5.8 Lines and sub-lines found in image1 with T=200
(a) Lines (b) Sub-lines

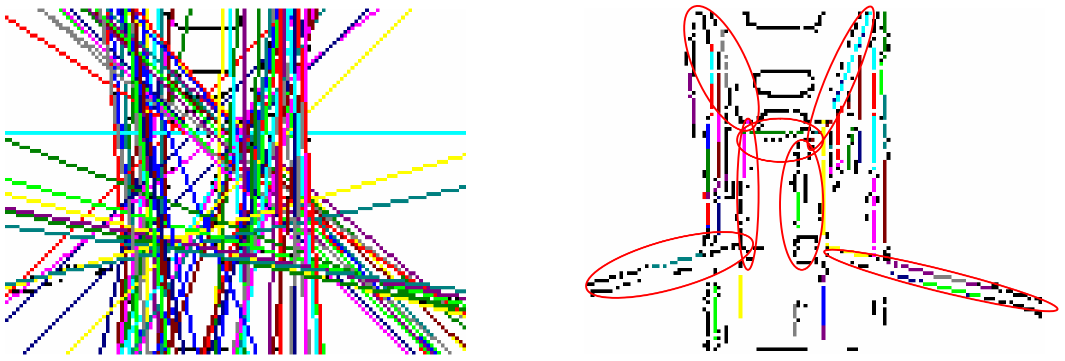


Figure 5.9 Lines and sub-lines found in image1 with T=250
(a) Lines (b) Sub-lines

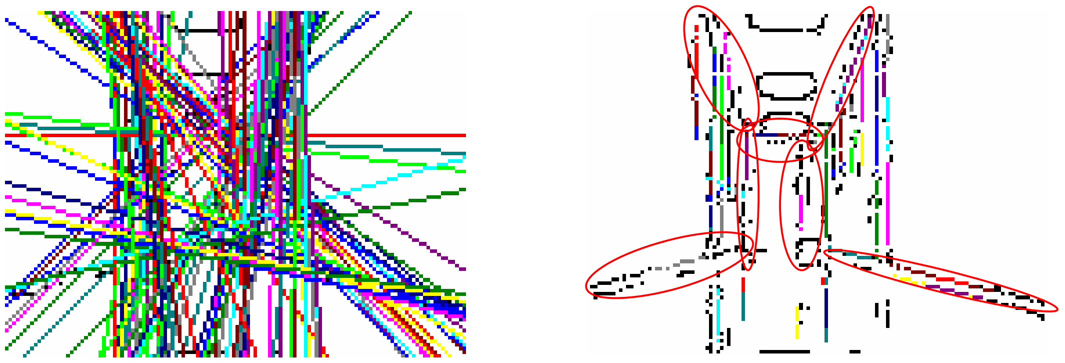


Figure 5. 10 Lines and sub-lines found in image1 with T=300
(a) Lines (b) Sub-lines

From table 5.1, the number of peaks found with T=250, 289, is higher than the 202 found using T=200. Also, the number of peaks after butterfly filtering, and the number of actual sub-lines found is higher in the case for T = 250. T = 250 will therefore take up more memory than T = 200. This means there is an increase in storage requirement for T= 250, and no corresponding increase in useful information as far as the vision-for-navigation system is concerned. This also applies for the increase from T= 250 to T = 300.

Although there were variations to the times taken for peak detection and finding sub-lines, the all variations up to the highest variation of 0.016s for both are not significant as established in *4.4.2.1 Errors in Time Measurement*. There is in fact an unexpected time taken to find sub-lines of 0 for T = 200. It is safest to conclude that there is no significant conclusion to be drawn regarding processing times. This also applies to the results for images 2 and 3.

Target Number of Peaks	**100**	**150**	**200**	**250**	**300**

Hough Transform time (milliseconds)	0.094	0.109	0.109	0.109	0.187
Peak Detection Time (milliseconds)	0.031	0.031	0.031	0.031	0.047
Number of Peaks found	121	182	202	289	364
BF Application Time (milliseconds)	0.016	0.015	0.016	0.016	0.015
Number of Peaks after Butterfly filtering	29	30	37	49	59
Number of valid lines with sub-lines	23	24	26	29	33
Number of sub-lines found	43	40	46	49	54
Time to find sub-lines (milliseconds)	0.015	0.015	0	0.016	0.032

Table 5.1 Processes information for image1 with various targets for number of edges

So from the situation with image1, 200 is the best value for T of the 4 values considered.

For image2, there is no improvement on the navigationally useful information beyond T = 250 as figures 5.11 to 5.16 shown.

There is, however, considerable increase to the number of lines found after application of the butterfly filter, and the actual sub-lines found from T = 250 to T = 300. This means considerably more memory is used when T is increased to T =300.

It may be concluded therefore, that for the purpose of this work, 250 is the best value of T for image2.

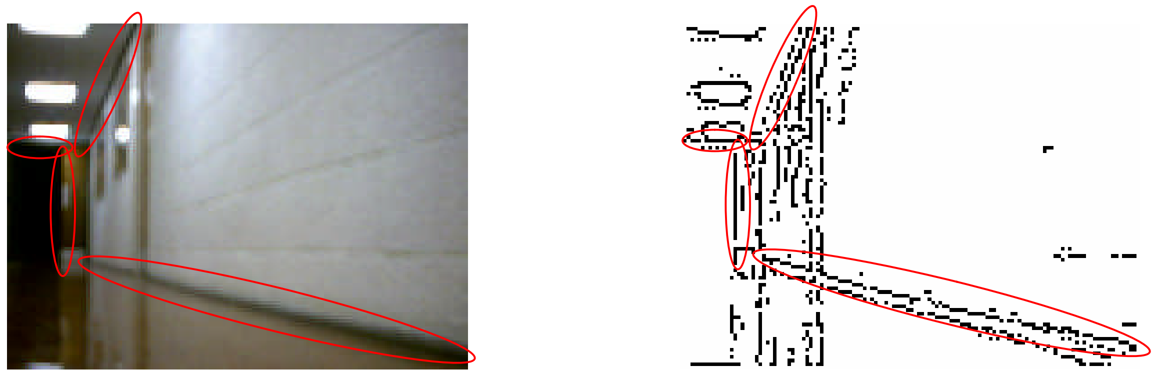


Figure 5.11 Navigation critical lines in a typical image – image2
(a) A typical image (b) Thinned version of image2

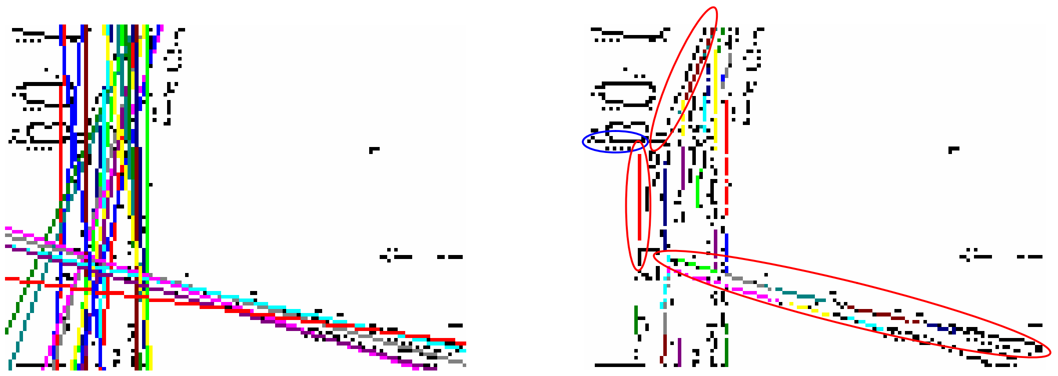


Figure 5.12 Lines and sub-lines found in image2 with T=100
(a) Lines (b) Sub-lines

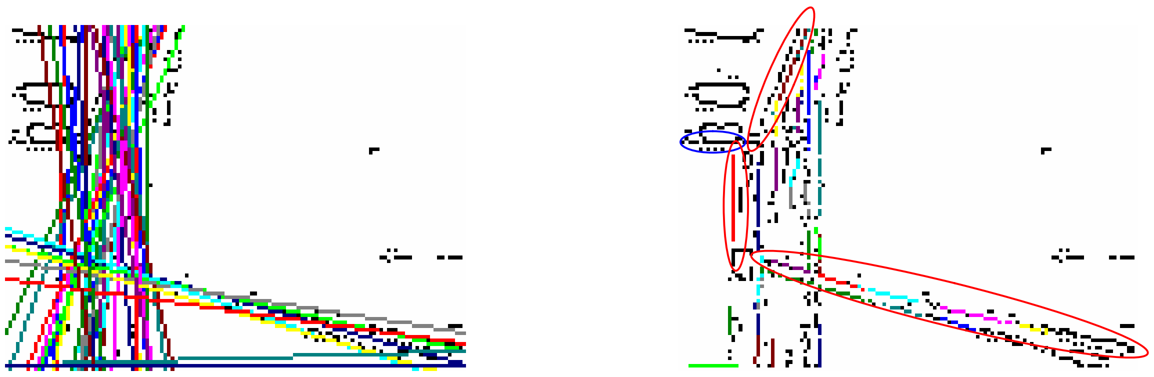


Figure 5.13 Lines and sub-lines found in image2 with T=150
(a) Lines (b) Sub-lines

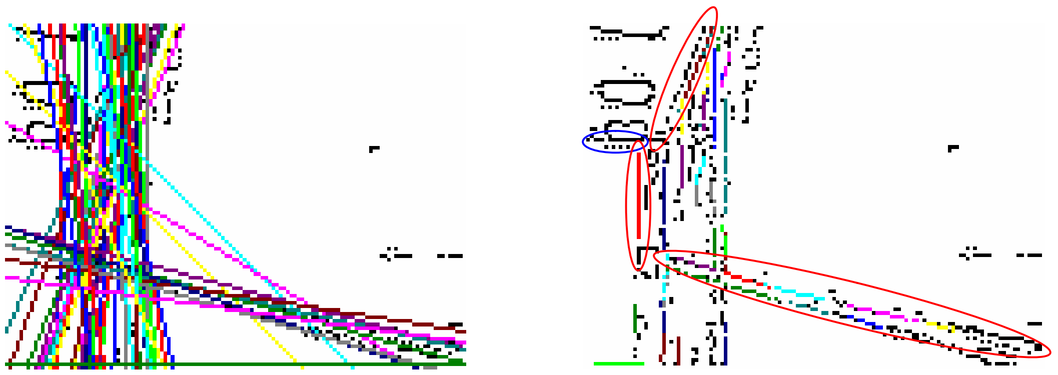


Figure 5.14 Lines and sub-lines found in image2 with T=200
(a) Lines (b) Sub-lines

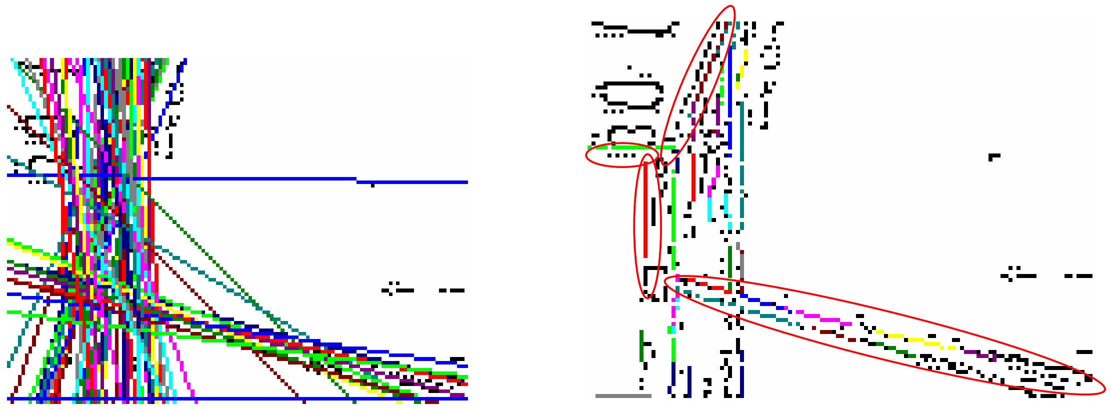


Figure 5. 15 Lines and sub-lines found in image2 with T=250
(a) Lines (b) Sub-lines

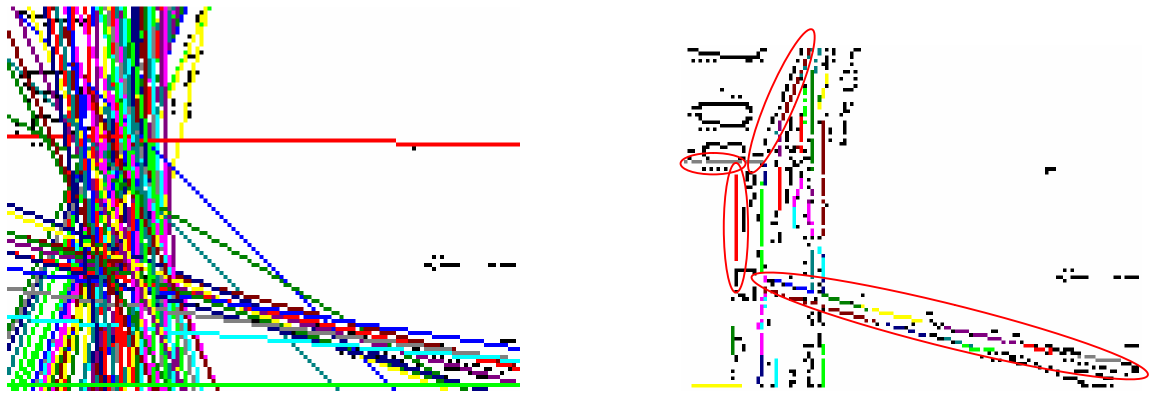


Figure 5.16 Lines and sub-lines found in image2 with T=300
(a) Lines (b) Sub-lines

Target number of Peaks	**100**	**150**	**200**	**250**	**300**
Hough Transform time (milliseconds)	0.094	0.078	0.094	0.094	0.094
Peak Detection Time (milliseconds)	0.031	0.031	0.031	0.031	0.031
Number of Peaks found	104	158	220	262	332
BF Application Time (milliseconds)	0.016	0.016	0.016	0.0160	0.032
Number of Peaks after Butterfly filtering	21	29	33	37	45
Number of valid lines with sub-lines	21	28	28	29	31
Number of sub-lines found	36	43	43	44	47
Time to find sub-lines (milliseconds)	0.015	0.016	0.016	0	0.016

Table 5.2 Processes information for image2 with various targets for number of edges

For image3, results are shown in figures 5.17 to 5.22. Again there was no improvement beyond 200, even though there are considerably more peaks and sub-lines as the values of T increased as table 5.3 shows. For this image, 200 is the best value to choose.

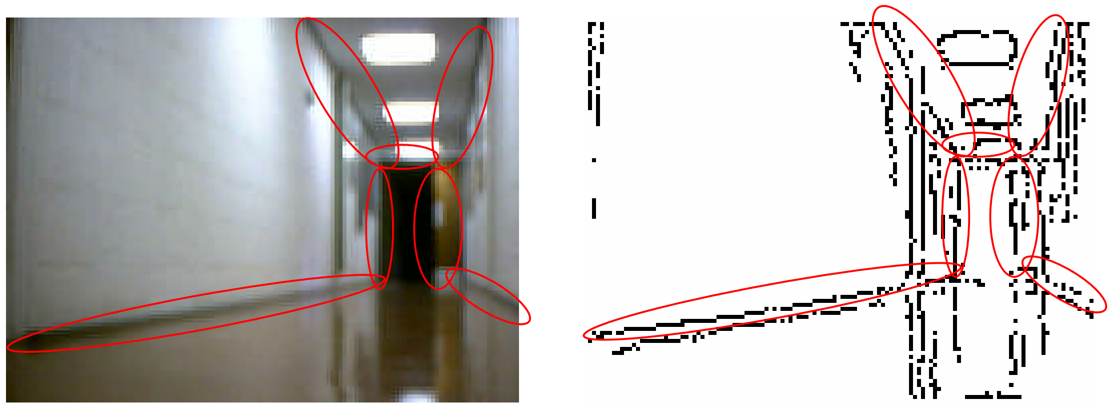


Figure 5.17 Navigation critical lines in a typical image – image3
(a) A typical image (b) Thinned version of image3

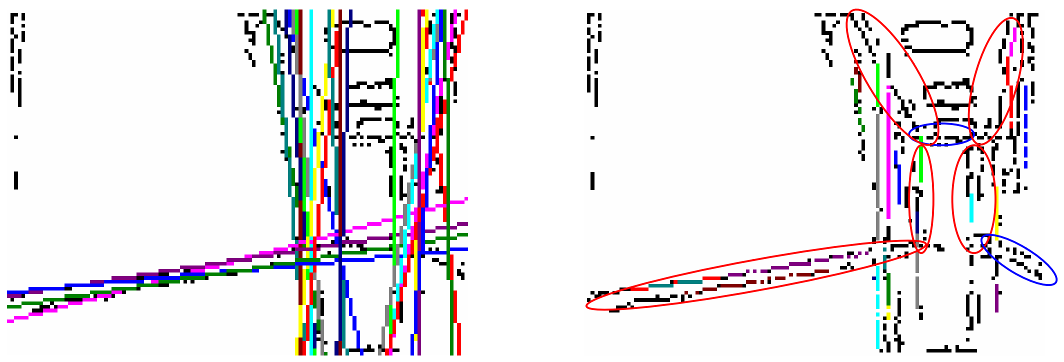


Figure 5.18 Lines and sub-lines found in image3 with T=100
(a) Lines (b) Sub-lines

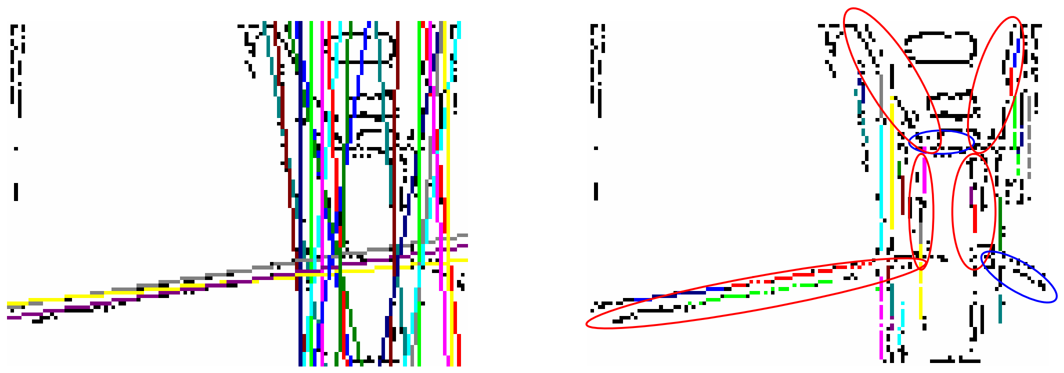


Figure 5.19 Lines and sub-lines found in image3 with T=150
(a) Lines (b) Sub-lines

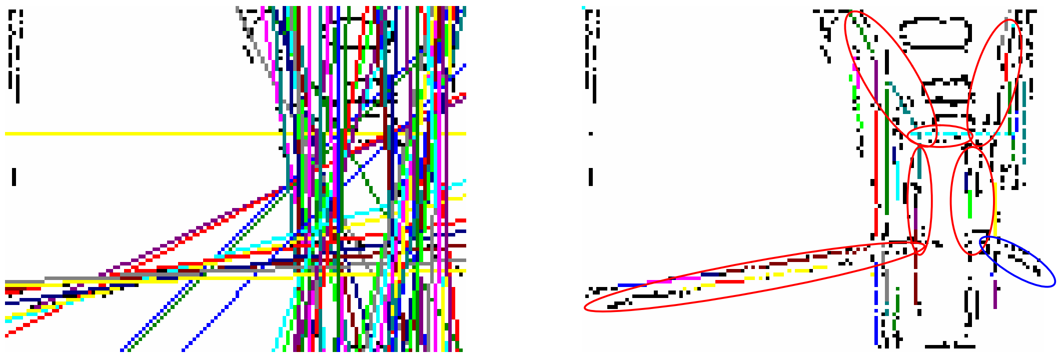


Figure 5.20 Lines and sub-lines found in image3 with T=200
(a) Lines (b) Sub-lines

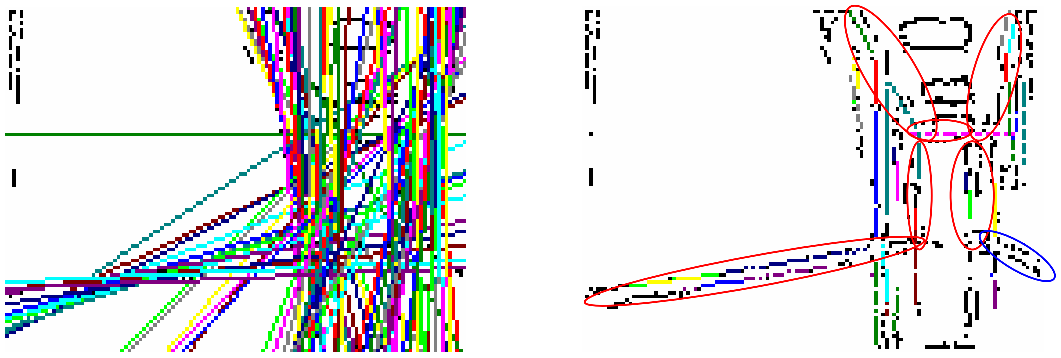


Figure 5.21 Lines and sub-lines found in image3 with T=250
(a) Lines (b) Sub-lines

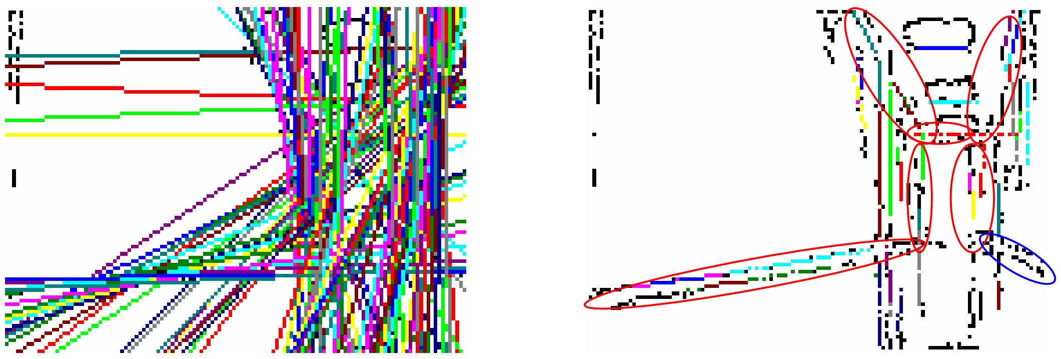


Figure 5.22 Lines and sub-lines found in image3 with T=300
(a) Lines (b) Sub-lines

Target number of Peaks	**100**	**150**	**200**	**250**	**300**
Hough Transform time (milliseconds)	0.094	0.109	0.110	0.109	0.187
Peak Detection Time (milliseconds)	0.047	0.015	0.0310	0.0310	0.047
Number of Peaks found	101	163	239	289	299
BF Application Time (milliseconds)	0.016	0.015	0.016	0.016	0.015
Number of Peaks after Butterfly filtering	18	25	38	49	49
Number of valid lines with sub-lines	17	23	29	29	30

Number of sub-lines found	26	32	40	49	40
Time to find sub-lines (milliseconds)	0	0.015	0	0.016	0

Table 5.3 Processes information for image3 with various targets for number of edges

In summary, from studying the results from typical images, it was concluded that 250 is the best value of T to use as it enabled detection of all the features that should reasonable be expected without taking up memory space unnecessarily.

5.1.6.2 Threshold Determination

The first stage in peak detection is determination of a threshold. As discussed in *1.1.6 Peak Detection*, this work uses a simple algorithm to work out the most appropriate threshold to employ given a target number of peaks. The target number of peaks was set at 250 after studying the results for various typical images for different values of T.

Let m be the lowest entry in accumulator array and let M be the highest entry. Let C_i be defined as the number of accumulator array entries with value i after application of the Hough transform algorithm, $m \le i \le M$.

The approach involves doing a cumulative sum, S_i say, where

$$S_i = \sum_{j=m}^{i} C_j \qquad . \qquad . \qquad . \qquad \text{Equation 5.2}$$

i.e., S_i is the sum of all accumulator array entries from the least entry to the i^{th} entry. S_i is then checked against N', the target number of non-peaks given by $N' = (N - T')$ where N = total number of accumulator array entries and T' = target number of significant elements.

While $S_i < N'$, the next value of S_i is determined.

If a point is reached where $S_i = N'$, i is taken as the threshold. Otherwise, as soon as $S_i > N'$, the last value of accumulator array entry before the current one, $i - 1$, is taken as threshold. This ensures that at least the required number of peaks is returned. Where $S_i = N'$, the number of peaks required equals the targeted number of peaks, T'. Where threshold is taken as soon as $S_i > N'$, the number of peaks returned will be greater than T' but less than $(T' + C_{i-1})$.

5.1.6.3 Application of the Butterfly Filter

The butterfly filter first proposed by (Boyce et al, 1987) is commonly used to enhance actual peaks and suppress false peaks within small neighbourhoods of them. The reduced filter was also proposed by (Boyce et al 1987) to be used for applications requiring low processing times. The reduced butterfly filter is used in this work as it yields fairly good results without requiring as much algebra processing as the full filter. In this work, as part of the scheme presented in *5.1.6 Peak Detection*, the butterfly filter is only applied to elements marked as peaks from the threshold application stage described in *5.1.6.1 Determination of Target Number of Peaks* above. The reduced butterfly filter is shown in figure 2.2

The reduced butterfly filter is 3 x 3 in size and at the point of application, the element of the array under consideration must have rows above it, below it, to the left of it and to the right of it. So if the accumulator array is designed within the theoretical range of [0°,180°) for θ and $\left[-\sqrt{(h^2+w^2)}, \sqrt{(h^2+w^2)}\right)$ for ρ as discussed in *5.1.3 Range of the Hough Transform Function*, a butterfly filtered array is obtained which only has meaningful values in the range [1°,179°) for θ and $\left[(-\sqrt{(h^2+w^2)}+1), (\sqrt{(h^2+w^2)}-1)\right)$ for ρ. The butterfly filter cannot yield meaningful results for the first and last rows, and the first and last columns of the accumulator array. This is particular undesirable because entries for $\theta = 0^0$ and $\theta = 179^0$, i.e. most representations of vertical lines are lost.

Furthermore, step 4 described in *5.1.6.5 Determination of Local Maxima*, requires finding local maxima within 5 x 5 neighbourhoods. This means that maxima within the second and second-to-the-last rows and the third and third-to-the-last columns of the accumulator array are also lost. This further reduces the range from which peaks can be detected to [3°, 177°) for θ and $\left[(-\sqrt{(h^2+w^2)}+3), (\sqrt{(h^2+w^2)}-3)\right)$ for ρ. There is no straightforward way of recovering the information lost due to shrinkage of the ρ axis. To get around this for the θ axis, it is necessary to either find a way to wrap from the 180° end of the accumulator array back to the 0°

when applying the butterfly filter and choosing maxima, or simply extend the array a little as the same lines begin to repeat at 180° and beyond. The same lines show up at 0° and 180°, and at 2° and 182° for example. What changes is the sign of ρ. This work extends the array to 185° so that 180° can make up for 0° lost to application of the butterfly filter, and 181° and 182° can make up for 1° and 2° lost to the criteria that a genuine peak must be a maxima in a 5 x 5 neighbourhood it is at the centre of. 183° and 184° are needed to complete 5 x 5 neighbourhoods for entries with θ values of 182°, and 185° is needed to provide a 3 x 3 region for application of the butterfly filter for entries with θ values of 184°. This means there are 186 columns in the accumulator array.

The algorithm applied follows. Note that 186° is used for the width of the array as it is the range of values for θ used as explained in the paragraph above, and 160, the height of the array is the range of the Hough transform function as discussed in *5.1.3 Range of the Hough Transform Function*. Note also, that steps 6 and 7 merely set the values for the first and last rows and the first and last columns to 0 because the filter cannot be applied to them as explained earlier in this section.

```
1.  Let $A$ be the accumulator array with $w = 186$ as its width, and $h = 160$ as its
    height, and $B$ the butterfly filtered accumulator array with the same
    dimensions.
2.  For all accumulator array rows $j$, $j$ running from 1 to $h - 2$ do 3
3.  For each column $i$ of current row $j$, $i$ running from 1 to $w - 2$ do 4
4.  If $A_{i,j}$ has been marked as a peak do 5
5.          (i)   $B_1 = -1 \times A_{i-1,j-1} - 4 \times A_{i,j-1} - 1 \times A_{i+1,j-1}$
            (ii)  $B_2 = 2 \times A_{i-1,j} + 8 \times A_{i,j} + 2 \times A_{i+1,j}$
            (iii) $B_3 = -1 \times A_{i-1,j+1} - 4 \times A_{i,j+1} - 1 \times A_{i+1,j+1}$
            (iv)  $B_{i,j} = B_1 + B_2 + B_3$

6.  Set top and bottom rows to 0, i.e,
    For all $i$, $i$ running from 0 to $w - 1$
        (i) $B_{i,0} = 0$
        (ii) $B_{i,h-1} = 0$.
```

7. Set side columns to 0

For all j from 0 to $h-1$

(i) $B_{0,j} = 0$

(ii) $B_{w-1,j} = 0$

5.1.6.5 Determination of Local Maxima

As part of efforts to minimise the detection of multiple peaks in parameter space due to a single line from image space, peaks are eliminated if their butterfly filtered value is not higher than that of all other entries within a 5 x 5 pixel neighbourhood surrounding them. Peaks which are local maxima within the 5 x 5 neighbourhood surrounding them are the target significant lines from the original image and they are processed further.

5.2 Post Processing

Section *5.1 Implementation of the Hough Transform* presented implementation of the Hough transform in this work. This section discusses higher level processes which determine the features that lines found represent. This includes discussions about reversing the Hough transform to acquire equations for significant lines in the original image, finding actual sub-lines from the lines (of undefined lengths) found, vanishing point recognition, corridor recognition, and door recognition.

5.2.1 Determination of Actual Lines

When the process of peak detection described in *5.1.6 Peak Detection* has been completed, the θ and ρ values of the peaks determined define the most significant lines from the original image. The equations of the lines required can be obtained using equations 2.7, 2.8 and 2.9 discussed in *2.1.8.1 Line Equation Reconstruction*. However the lengths of the lines they came from are still not defined. Endpoints need to be determined for genuine lines, and ‘false’ lines which emerged by accumulating votes from random points by ‘coincidence’ need to be eliminated. Various approaches exist for determination of end points as discussed in *2.1.8.2 Endpoints and Length Determination* including approaches discussed in (Leavers 1992) and (Low 1991). One developed by this work is to track all points which contributed to each of the peaks found, and find out if they make up any valid sub-lines. Here a sub-line, S, say, for a line L, can be seen as a shorter line that is contained in L, i.e., all points that make up S also contribute to making up L. A valid sub-line is a sub-line which meets these further criteria:

1. It must reach a certain minimum number, $L_{\min}$, of pixels in length
2. It must have a minimum separation of a certain number of pixels, $S_{\min}$, from any other line segment on the same infinite line, otherwise the two lines segments are labelled as one line segment.

Endpoints and length of sub-lines can then be worked out from the points that make them up.

Determination of values for the parameters $L_{\min}$ and $S_{\min}$ for these criteria is discussed in *5.2.1.1 Values of Parameters for Criteria for Valid Sub-lines* which follows.

The algorithm for finding valid sub-lines, the sub-lines finding algorithm, has two main components which are:

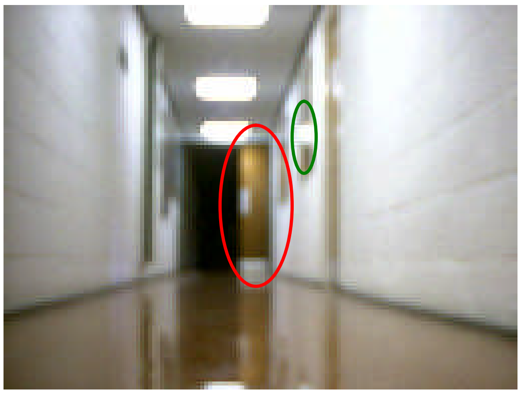
1. Determine which points contributed to each line
2. Find the number of points on each line, noting which ones have $L_{\min}$ points or more, and are at least $S_{\min}$ pixels away from other points or lines.

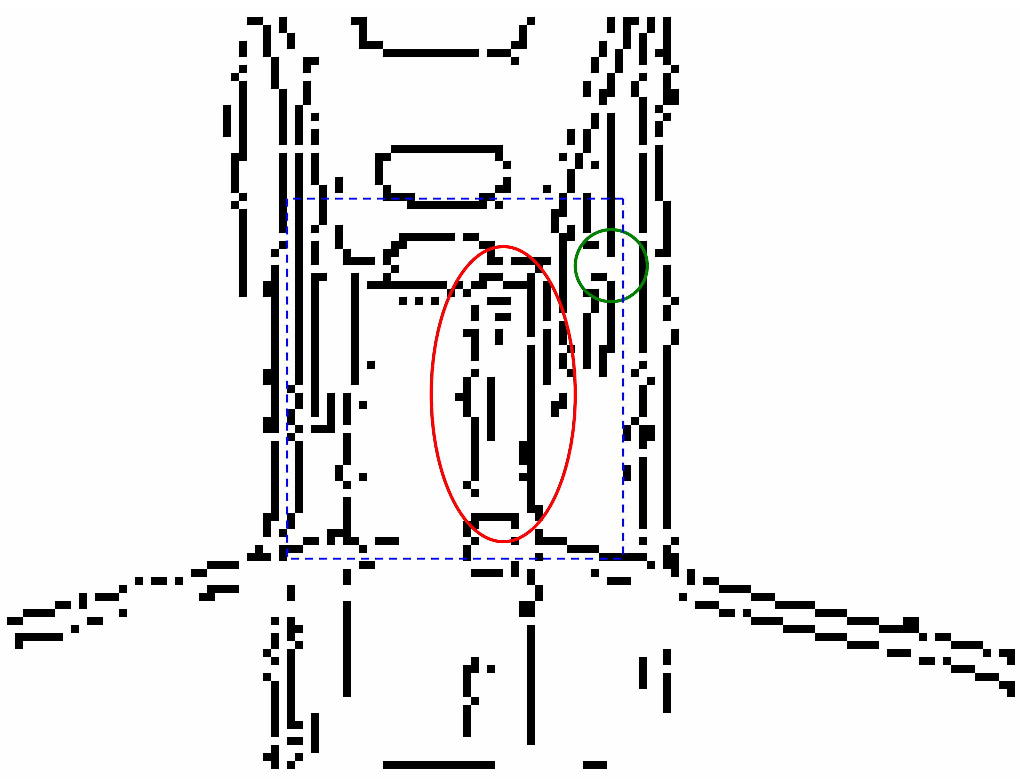
They are given in *5.2.1.2 Assigning Contributing Points to Sub-line*, and *5.2.1.3 Selection of Valid Sub-Lines*. *5.2.1.4 Sample Result of Sub-Lines Determination Algorithm* shows results for these two components for a typical image. An algorithm for determining endpoints and length for each valid sub-line is then discussed in *5.2.1.4 Endpoints and Length Determination*.

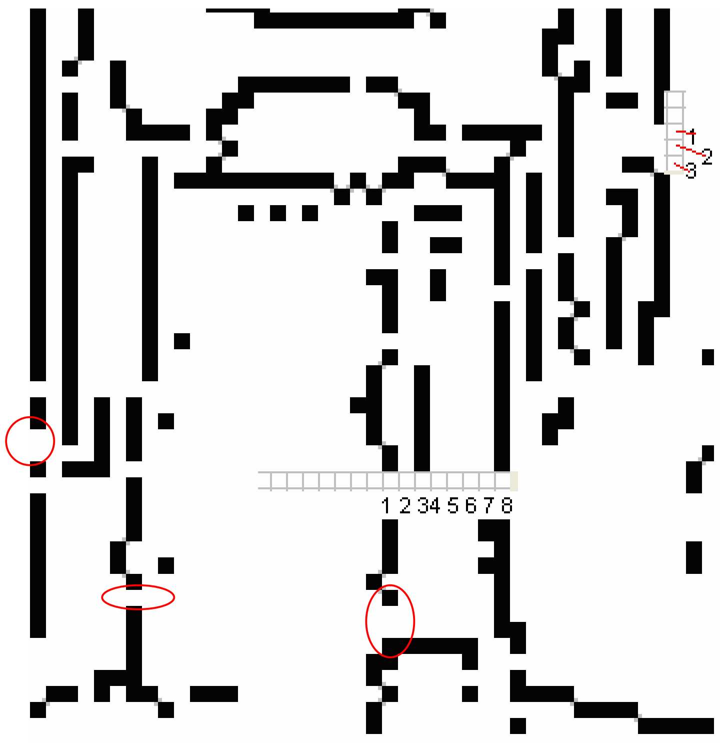
5.2.1.1 Values of Parameters for Criteria for Valid Sub-lines

The minimum distance for sub-lines, $L_{\min}$, was set at 8 because from studying typical images, it is a reasonable general minimum length for significant features in an image. Consider the images in figure 5.23. Figure 5.23a shows a typical image of a corridor magnified 2 times, and figure 5.23b shows a thinned version of it magnified 4 times for clarity. Figure 5.23c shows the area enclosed by the blue dashed box of 5.23b further magnified 2 times. For the purpose of this work, lengths such as the width of the brown area circled in red on the right of the door are about the smallest lengths of features that might be significant. Any feature shorter than that is probably too insignificant from that distance. As the robot moves nearer to objects, those objects of-course become larger, and have a higher chance of getting detected if they are important. The width of the brown bit is represented by a length of approximately 8 pixels as can be estimated by closely examining figure 5.23b and 5.23c.

$S_{\min}$ was chosen in a similar manner. Within the area surrounded by the green cycle, there is what would be to a human observer, a separation between two features. In 5.23b, the two features appear as one continuous line. The Hough transform would pick them up as 1 line as depicted in figure 5.23d. The separation between them is 3 pixels, as can be seen by studying 5.23b and 5.23c. It can be argued that a separation of about that much is about the least that should reasonably be expected to be identifiable. Any separation less than that is likely to be a crack in a genuine line. Examples of such are circled in red in figure 5.23c.






1
2
3
1 2 3 4 5 6 7 8

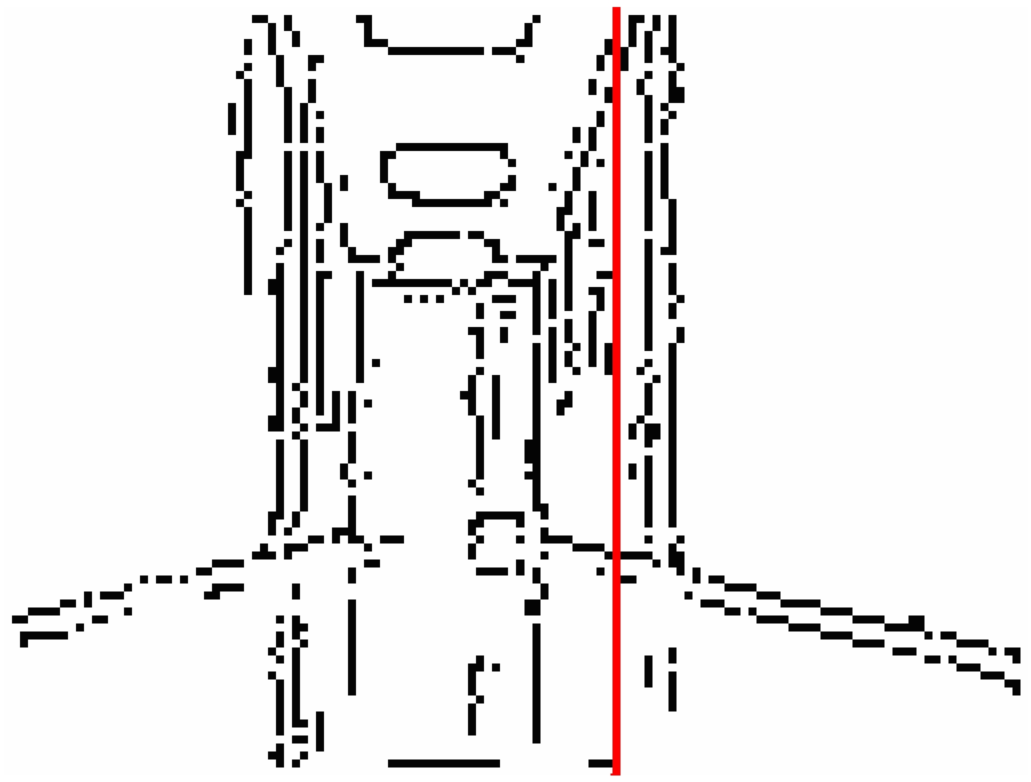


Figure 5.23 Parameters for selection criteria for valid sub-lines
(a) A typical image showing smallest significant length of line segment and separation between 2 line segments on the same line (b) Thinned version of figure 5.23a image (c) Enlargement of the area of *figure (b)* image enclosed in blue dashed rectangle (d) Single line found by the HT consisting of multiple sub-lines

5.2.1.2 Assigning Contributing Points to Sub-lines

The first component of the sub-line determination algorithm of *5.2.1 Determination of Actual Lines* is presented here. Its purpose is to determine which sub-line each point belongs to. The steps taken follow:

1. Set up a variable, LN (line number) say, to count the number of lines found and initialise it to 0

2. Set up an array LI (line-in) of size CP (contributing points) to hold information about which line each point in the significant line found from the processes discussed to this point is in. CP is the number of points which contributed to that line.

3. Set up two types of pivot indices. The first, LP, the line identification pivot, which tracks the first point of the current line, and the second, DP, the distance pivot, which tracks the point on the line that is currently being compared with other points to see if they are on the line. Initialise both P and DP to 0

4. Set up a distance threshold, dT. This is the smallest distance between consecutive contributing points which implies that the two points are on separate sub-lines (on the same infinite line). In this work, this is set to 3 as discussed earlier in this section

5. For all points p which contributed to the infinite line under consideration, initialise the LI array to -1 (or any number which will not arise naturally)

6. While LP, the pivot, is less than CP, the number of points which contributed to the line under consideration, do 7 to 16

7. If $LI_{LP} = -1$, i.e. if the line that point LP is in has not yet been determined, do 8 else do 17

8. Assign the line-in array element for the current pivot LI_{LP} to LN (a fresh line ID). Increase LN by 1 to make it ready for the next fresh line. Set DP, the distance pivot to LP the line label pivot

9. Set p to $LP+1$

10. If $LI_p = -1$, i.e. if point p has not been processed already, do 11 else do 15

11. Obtain n_{DP}, the index of DP the distance pivot in the original image, using $n_{DP} = CPA_{DP}$. (CPA is the array which holds indices of contributing points for lines, i.e. points from image space which contributed to each line. It is set up during implementation of the Hough transform). Determine x_{DP} and y_{DP}, the coordinates of the point in image space using:

$$x_{DP} = (n_{DP}\%128 - 64)$$

$$y_{DP} = (47 - n_{DP}/128)$$

12. Obtain n_p, x_p and y_p in a similar way for p, the point being checked against DP using:

$$n_p = CPA_p$$

$$x_p = (n_p \%128 - 64)$$

$$y_p = (47 - n_p / 128)$$

13. Determine the distance, d, between the two points using:

$$d = \sqrt{(x_{DP} - x_p)^2 + (y_{DP} - y_p)^2}$$

14. If $d < dT$ place p on the same sub-line as line pivot LP,

$$LI_p = LI_{LP}$$

and make p the distance pivot

$$DP = p$$

15. Increase p by 1
16. Go back to 10 if p has not reached CP
17. Increase pivot LP by 1 and go back to 6

5.2.1.3 Selection of Valid Sub-Lines

To check for 'validity' of sub-lines, the second component of the sub-line determination algorithm of *5.2.1 Determination of Actual Lines*, the following steps are taken.

1. Set up an array LL (line length) of size LN to hold the length of each possible sub-line in each significant line found from the processes discussed to this point. LN is number of lines (groups of points with 3 or more pixels separation from all other points) found.

2. Determine the length LL_l for all lines l, $0 \le l < LN$, by counting the number of points which make it up.

3. For all lines which have 8 points or more, i.e., $LL \ge 8$, assign a unique ID, note the number of points which make up the line, and note the ID numbers of all the points which make up the line

5.2.1.4 Sample Result of Sub-Lines Determination Algorithm

The images in figure 5.24 which follows illustrate the results of the sub-lines determination algorithm. Figure 5.24a shows a typical thinned image, 5.24b shows the lines found after application of the Hough transform and post processing to the point of butterfly filter application. There were 49 of them. Figure 5.24c shows the sub-lines found by this algorithm. 40 sub-lines were found altogether. Figure 5.24c was set up by plotting individual points that contributed to each sub-line. The first sub-line found on a line segment is coloured red. If there is a second sub-line, it is coloured blue, and if there is a third one, it is coloured green. (No lines encountered in the course of the work had more than 3 sub-lines.)

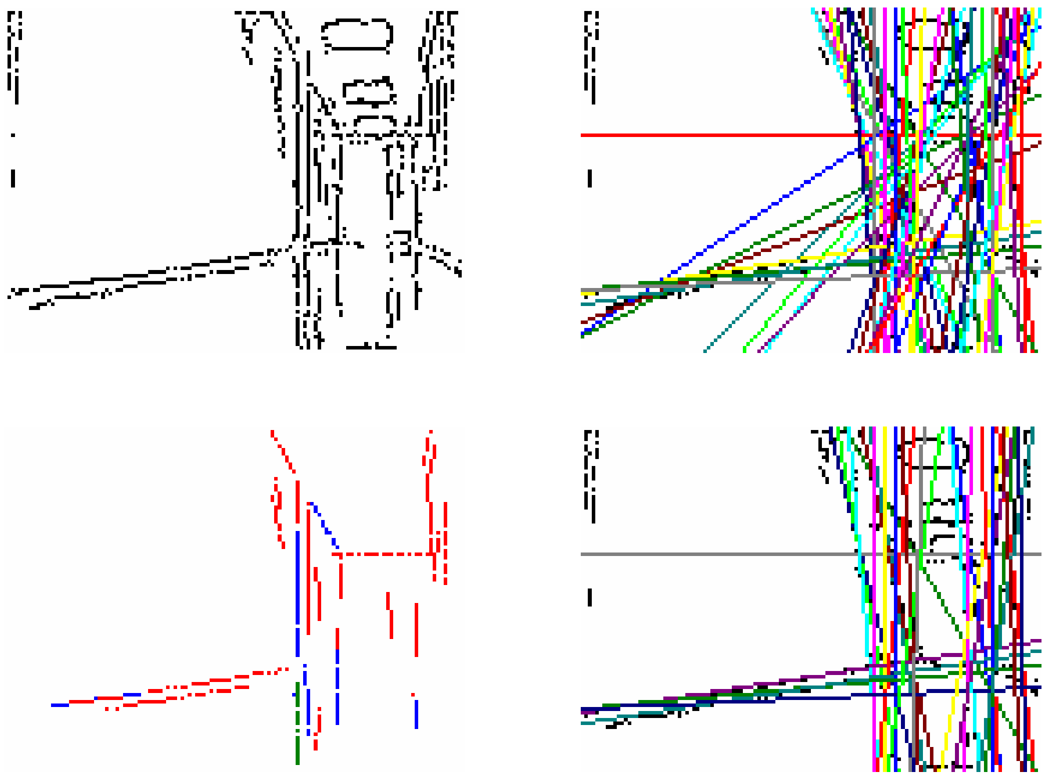


Figure 5.24 Result of sub-line detection algorithm
(a) Typical thinned image input to algorithm 5.8 for finding sub-lines (b) Lines found with the Hough transform (c) Sub-lines found (d) Lines from which sub-lines were found

5.24d shows the lines from which the sub-lines were extracted. There were 30 of them. Infinite lines which do not have any suitable sub-lines are dismissed as false recognitions. In this example, 19 lines were dismissed for this reason. *Appendix B* contains further details about the number of sub-lines found in each line, and the points which made up each sub-line.

5.2.1.5 Endpoints and Length Determination

Up till this point, valid sub-lines have been found and are defined by a unique identification number, the line they came from and the identification numbers (or indices) of their contributing points CP, i.e. the points that make them up. A sample of these details for the image in figure 5.24 is given in *appendix B*. Endpoints can now be determined.

The scheme which follows was used by this work. At the core of it, it involves:

1. Pick up the index of the first point of a sub-line and use it as a pivot point:

   $x_0 = ID\%128$

   $y_0 = ID/128$

   $x_{pivot} = x_0$

   $y_{pivot} = y_0$

2. Initially set the first point to be the farthest point on both sides of the line

   $mxDistIndex = ID$

   $mnDistIndex = ID$

   and the maximum and minimum distance to 0

   $mxDist = 0$

   $mnDist = 0$

3. For all other points i which contributed to the sub-image, do 4 to 8

4. If the gradient of the line is less than or equal to 1 do 5 and jump to 7 else do 6

5. Calculate the horizontal distance from the pivot point, i.e. the first point, to point i using

   $dist = x_{pivot} - x_i$

6. Calculate the vertical distance from the pivot point, i.e. the first point, to point i using

   $dist = y_{pivot} - y_i$

7. If the distance is higher than the currently stored maximum distance, use it to replace the currently stored maximum distance, and store its index as the index of the maximum distance

   if $dist > mxDist$ then $mxDist = dist$; $mxDistIndex = index$

8. If the distance is lower than the currently stored minimum distance, use it to replace the currently stored minimum distance, and store its index as the index of the minimum distance

   if $dist < mnDist$ then $mnDist = dist$; $mnDistIndex = index$

Steps 1 to 8 are done for each sub-line and result in the indices of the extreme points of the sub-line being stored in $mxDistIndex$ and $mnDistIndex$. The sequence of steps finds the highest and the lowest distance from the first point. If the first point is itself at one of the extreme positions of the sub-line, then one of the distances will be 0, otherwise, one will be positive and the other will be negative. Step 4 is necessary to ensure that the distance is calculated in a direction such that the kind of distance – vertical or horizontal – chosen will result in the most noticeable distances along the line. There is no point working out horizontal distance between points on a vertical or near vertical line for example as they will all be about the same.

Once the indices of the points are known, their x and y coordinates can be worked out using the integer operations

$$x_{mxDist} = mxDistIndex\%128 \qquad . \qquad . \qquad . \qquad \text{Equation 5.3a}$$

$$y_{mxDist} = mxDistIndex/128 \qquad . \qquad . \qquad . \qquad \text{Equation 5.3b}$$

$$x_{mnDist} = mnDistIndex\%128 \qquad . \qquad . \qquad . \qquad \text{Equation 5.3c}$$

$$y_{mnDist} = mnDistIndex/128 \qquad . \qquad . \qquad . \qquad \text{Equation 5.3d}$$

The length, L of the sub-line can also be worked out using

$$L = \sqrt{\left(x_{mxDist} - x_{mnDist}\right)^2 + \left(y_{mxDist} - y_{mnDist}\right)^2} \qquad . \qquad . \qquad . \qquad \text{Equation 5.4}$$

5.2.3 Vanishing Point Estimation

Where it can be found, the primary vanishing point in an image can facilitate analysis of the image. An early step taken towards high level feature recognition in this work is to try and find the primary vanishing point in the image. An algorithm has been developed to do this, which takes advantage of certain characteristics of vanishing points, and the nature of the images encountered in this work. This algorithm is based on likelihoods, and although it has worked in most cases, it sometimes fails.

The algorithm involves determination of points of intersection for all pairs of significant lines found from the Hough transform. Because this work is done with images taken from a rectilinear corridor environment, in most images encountered, lines due to corridor-floor boundaries and corridor roof boundaries intersect at the vanishing point. This means that the vanishing point can be expected to have a few lines intersecting on it.

Suppose that (θ_i, ρ_i), i = 1 to n, n being the number of significant lines determined from the application of the Hough transform, is the set of significant lines resulting from application of the Hough transform and subsequent post-processing to the point of dismissing lines without valid sub-lines in them. This means significant lines found are

$$\rho_i = x\cos\theta_i + y\sin\theta_i, \; i = 1 \text{ to } n \quad . \quad . \quad . \text{ Equation 5.5}$$

Vanishing points are determined by finding points to which all pairs of lines appear to converge. This can be achieved by determining intersection points between every pair of lines. To simplify this, the set (m_i, c_i) of corresponding gradients and intercepts are found using equation 2.7, 2.8 and 2.9.

When intersection points have been found, they are clustered based on proximity to allow for errors in line determination and intersection point calculations. A note is made of the number of points contributing to each cluster, and of the pairs of lines which intersect at those points. This information is vital for later processing.

Each cluster of intersection points found is examined against the following criteria:

1. Whether or not it is on the horizon: For images in this work, the horizon will normally be around the horizontal midpoint of images as the camera view is set approximately parallel to the ground. Only points on or very close to the horizon are considered further.

2. Number of lines that intersect on point: Points are ranked according to how many lines intersect on them. If more than 10 clustered intersection points are found, only the 10 with the highest number of intersecting lines are considered. If the number of clustered intersection points is less than or equals to 10, then all points are considered for further processing.

   Each point is assigned a weight. 10 is the maximum weight and is assigned to the point with the highest number of intersecting lines, 9 is assigned to the point with the second highest number of intersecting lines, etc.

   Where two or more points have the same number of intersecting lines, they are assigned the same weight and subsequent points are assigned weights which reflect the number of points with a higher number of intersecting lines than them. If 3 points have the same the number of lines and are assigned a weight of 8, for example, the next point is assigned a weight of 5, not 7, as weights 6 and 7 would have been assigned but for the equality.

For criteria 3 to 5 below, a preliminary categorisation of significant lines found is done based on their θ values. This categorisation is done into the classes "vertical to the left of", "vertical to the right of", "horizontal above", "horizontal below", "slash through", "backslash through", and "vertical on". These categories are elaborated in table 5.4 which follows.

Class	**Θ Lower Bound**	**Θ Upper Bound**	**Θ Range**	**Minimum Distance from Point**	**Maximum Distance from Point**
Vertical to the left	-7 (or 173)	7	15	2	n/a
Vertical to	-7 (or	7	15	2	n/a

the right	173)				
Slash through	15	75	60°	n/a	2
Horizontal above/below	83	98	15	2	n/a
Backslash through	105	165	60°	n/a	2
Vertical on	-5 (or 175)	4	10	n/a	2

Table 5.4 Definitions for preliminary classes of lines

3. Whether or not there is an “X” on point: There are two parts to this criterion. One part is fulfilled if there is a “slash through” line on the point under consideration. If it is met, a weight of 1.5 is assigned. The second part is met if there is also a “backslash through” line on the point. A further 1.5 is assigned to the point if this is the case.

   For the purpose of step 3, “slash through” and “backslash through” are as defined in table 5.4.

4. Whether or not there is a “rectangle” around it: Is there a set of four (or more) significant lines which include:
   - one that is approximately “horizontal above” it (a weight of 1.25)
   - one that is approximately “horizontal below” it (a weight of 1.25)
   - one that is approximately “vertical to the left” of it (a weight of 1.25)
   - one that is approximately “vertical to the right” of it (a weight of 1.25)

   Horizontal and vertical to the left (or right) are as defined in table 5.4.

5. “Vertical on” it: If there is a vertical line passing right through or very near the point under consideration, 2.0 is subtracted from its weight.

The point that scores the highest against these criteria is returned as the vanishing point, provided the total of points it has scored is at least 13. If 13 is not reached by any point, there is not enough evidence to confidently point out the vanishing point.

All the parameters in this algorithm were decided after trying out the algorithm several times with several typical images. Figure 5.25 shows the winning point (-6, -16) from a sample run (by a red dot). The winning score was 13.75. Table 5.5 shows a breakdown of the scores for the top ten points.

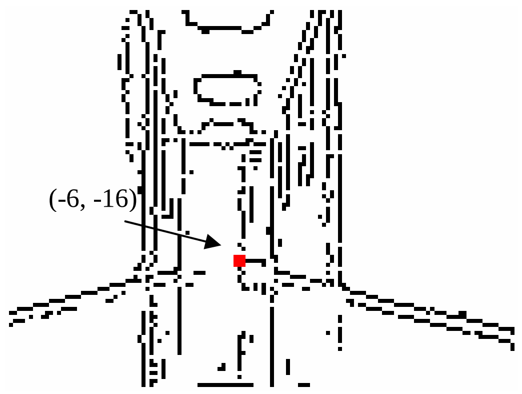


Figure 5.25 Sample Vanishing points found

Point (x,y)	Criteria	Score	Cummulative Score
-18 -18	num of intersecting lines	10	10
	has slash	1.5	11.5
	verToRight	1.25	12.75
	verToLeft	1.25	14
	horAbove	1.25	15.25
	verLineOn	-2	13.25
-6 -16	num of intersecting lines	9	9
	has slash	1.5	10.5
	Backslash	1.5	12
	verToRight	1.25	13.25
	verToLeft	1.25	14.5
	horAbove	1.25	15.75
	verLineOn	-2	13.75
-31 -9	num of intersecting lines	8	8
	has slash	1.5	9.5
	verToRight	1.25	10.75
	horAbove	1.25	12
	verLineOn	-2	10
0 -16	num of intersecting lines	7	7
	has slash	1.5	8.5
	Backslash	1.5	10
	verToRight	1.25	11.25

	verToLeft	1.25	12.5
	horAbove	1.25	13.75
	verLineOn	-2	11.75
-4 -21	num of intersecting lines	7	7
	has slash	1.5	8.5
	Backslash	1.5	10
	verToRight	1.25	11.25
	verToLeft	1.25	12.5
	horAbove	1.25	13.75
	verLineOn	-2	11.75
-13 -17	num of intersecting lines	7	7
	has slash	1.5	8.5
	Backslash	1.5	10
	verToRight	1.25	11.25
	verToLeft	1.25	12.5
	horAbove	1.25	13.75
-32 -21	num of intersecting lines	7	7
	verToRight	1.25	8.25
	horAbove	1.25	9.5
	verLineOn	-2	7.5
-1 -15	num of intersecting lines	3	3
	has slash	1.5	4.5
	verToRight	1.25	5.75
	verToLeft	1.25	7
	horAbove	1.25	8.25
-21 -12	num of intersecting lines	3	3
	has slash	1.5	4.5
	verToRight	1.25	5.75
	verToLeft	1.25	7
	horAbove	1.25	8.25
	verLineOn	-2	6.25
-31 -16	num of intersecting lines	3	3
	verToRight	1.25	4.25
	horAbove	1.25	5.5
	verLineOn	-2	3.5

Table 5.5 Breakdown of scores for top 10 intersection points

Table 5.6 summarises other sample results.

Image ID	**Original Image**	**Pre-processed Image with Lines Found and** VP	**Estimated** $VP(x,y)$	VP **found** (x,y)	**Weight**

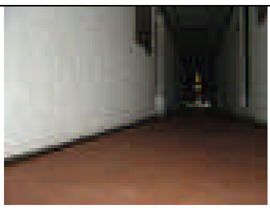
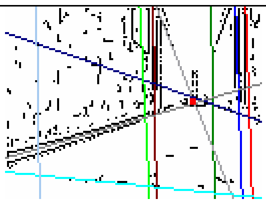
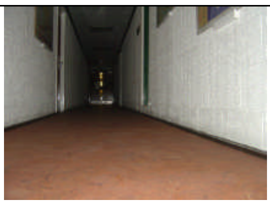
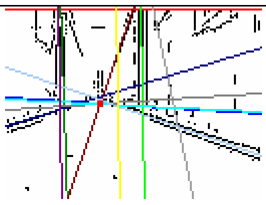
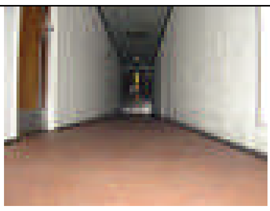
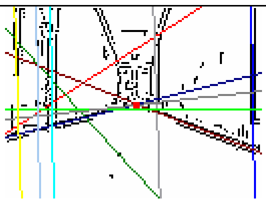
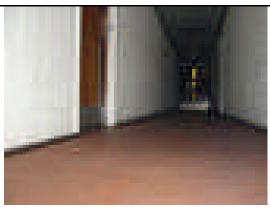
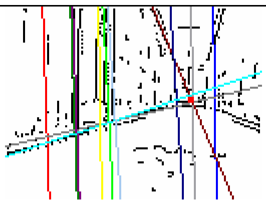
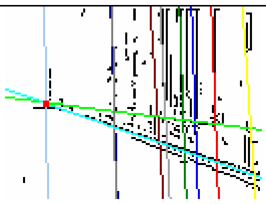
1648			30, 2	29, 1	16.75
1649			-17, 3	-17, 3	16.75
1650			-1,1	-3,0	14.25
1651			28,2	28, 2	13.5
1652			-43,-1	-44,0	12

Table 5.6 Some results from VP determination scheme

5.2.4 Category and Position Assignment to Lines

After the vanishing point has been determined, each line is assigned two attributes – type and displacement from vanishing point. Displacement consists of the magnitude of the distance from the vanishing point, and its direction or sign, i.e. whether or not the distance is positive or negative*. 5.2.4.1 Line Category*, *5.2.4.2 Magnitude of Distance for Line from Vanishing Point* and *5.2.4.3 Sign of Displacement of Lines from Vanishing Point* discuss these further.

5.2.4.1 Line Category

Possible types and the direction in which their distance from the vanishing point is measured are shown in table 5.7 which follows.

Category	**Description**	**Minimum**	**Maximum**	**Range**	**Direction of**

ID			Θ	Θ	**Size**	**Distance Measurement**
0	Vertical	│	-5 (or 175)	4	10	left to right
1	Vertical Backslash	╲	5	24	20	bottom left to top right
2	Backlash	╲	25	64	40	bottom left to top right
3	Horizontal Backslash	╲	65	84	20	bottom left to top right
4	Horizontal	—	85	94	10	bottom to top
5	Horizontal slash	╱	95	114	20	bottom right to top left
6	Slash	╱	115	154	40	bottom right to top left
7	Vertical Slash	╱	155	174 (or -6)	20	bottom right to top left

Table 5.7 Lines Categorisation

The column on the far right summarises directions of distance measurement for the various line types. A line passing through the vanishing point is at 0 position and other values are negative to positive along the direction of distance measurement shown. A vertical line on the left side of the image for example, would have a negative distance and a horizontal slash on the top left side of the image would have a positive value.

5.2.4.2 Magnitude of Distance of Line from Vanishing Point

The magnitude of the distance of a line from the vanishing point is the perpendicular distance between the vanishing point and the line.

As an illustration, figure 5.26 is a typical image showing the vanishing point, the red square labelled A with coordinates (x_{VP}, y_{VP}) and one of the lines found, shown as a red line and labelled BC.

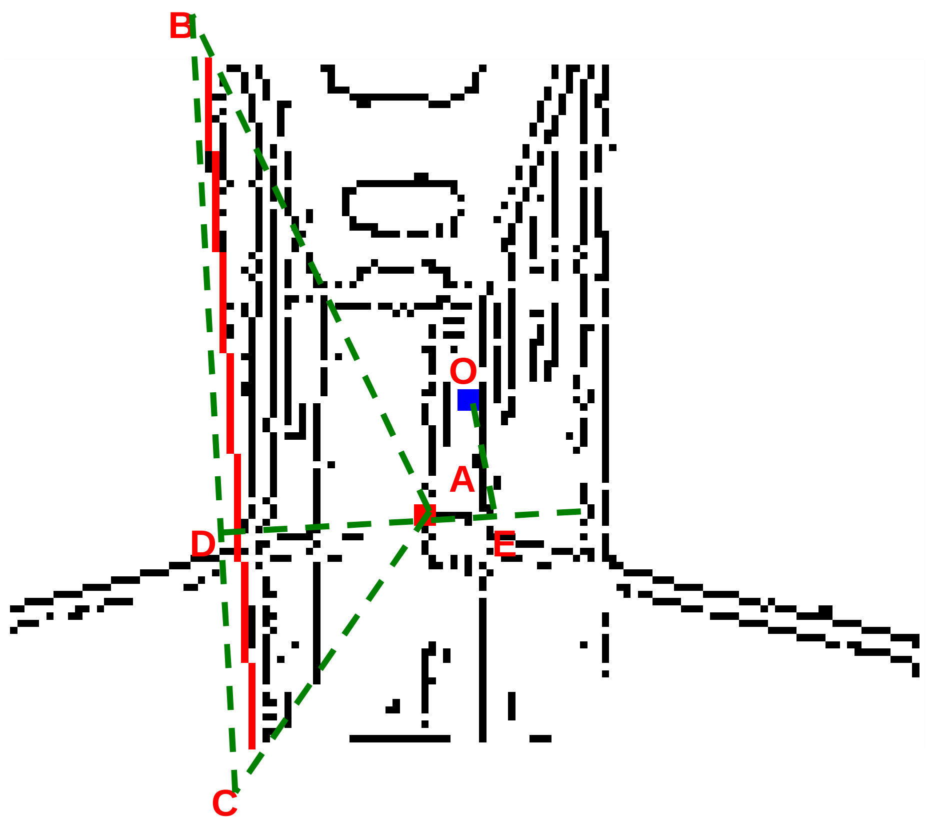


Figure 5.26 Determining magnitude of distance of line from vanishing point

The distance between points A and the point labelled D is the perpendicular distance required. It is determined by first calculating θ_{per} and ρ_{per}, the parameters which define the perpendicular line AD. The angle of AD, θ_{per} is simply θ for the line under consideration, line BC, plus 90° i.e.,

$$\theta_{per} = \theta + 90° \qquad . \qquad . \qquad . \qquad \text{Equation 5.6}$$

ρ_{per}, the distance of the perpendicular line AD to the origin O (shown as a blue square labelled O), i.e. the length OE, can be obtained using

$$\rho_{per} = x_{VP} \cos\theta_{per} + y_{VP} \sin\theta_{per} \qquad . \qquad . \qquad . \qquad \text{Equation 5.7}$$

because it is known that the vanishing point (x_{VP}, y_{VP}) lies on the line AD. θ_{per} and ρ_{per} can then be plugged into equations 2.8 and 2.9 to obtain the gradient m_{per} and the intercept c_{per} of the line AD, so the equation of AD

$$y = m_{per} x + c_{per} \qquad . \qquad . \qquad . \qquad \text{Equation 5.8}$$

The point of intersection of line AD and line BC can then be determine using

$$x_{intersection} = \frac{-\left(c_{BC} - c_{per}\right)}{m_{BC} - m_{per}} \qquad . \qquad . \qquad . \qquad \text{Equation 5.9}$$

and

$$y_{intersection} = m_{per} x_{intersection} + c_{per} \qquad . \qquad . \qquad . \qquad \text{Equation 5.10}$$

obtained by solving for $x_{intersection}$ and $y_{intersection}$ from the equation for line BC

$$y = m_{BC} x + c_{BC} \qquad . \qquad . \qquad . \qquad \text{Equation 5.11}$$

and equation 5.8.

The distance d between the intersection point ($x_{intersection}$, $y_{intersection}$) and the vanishing point (x_{VP}, y_{VP}) can then be worked out using

$$d = \sqrt{(x_{VP} - x_{intersection})^2 + (y_{VP} - y_{intersection})^2} \qquad . \qquad . \qquad \text{Equation 5.12}$$

and that is the distance required. The distance between all lines found and the vanishing point, if it is found, is evaluated in this way.

5.2.4.3 Sign of Displacement of Lines from Vanishing Point

To determine the sign of a line, the point in parameter space representing it is compared to the curve representing the vanishing point in parameter space. Where θ on the curve equals θ for the line, if the point lies on the negative side of the accumulator array relative to the curve, the sign of the line in the line categorisation scheme is negative and if it is on the positive side relative to the curve, the sign of the line is positive.

Take, for example, the lines found from the image in figure 5.27a after processing. The image, its pre-processed version, the lines and vanishing point found, and an enlargement of the part of it enclosed by the red dashed rectangle in numeric format are shown in figure 5.27a, 5.27b, 5.27c, and 5.27d respectively.

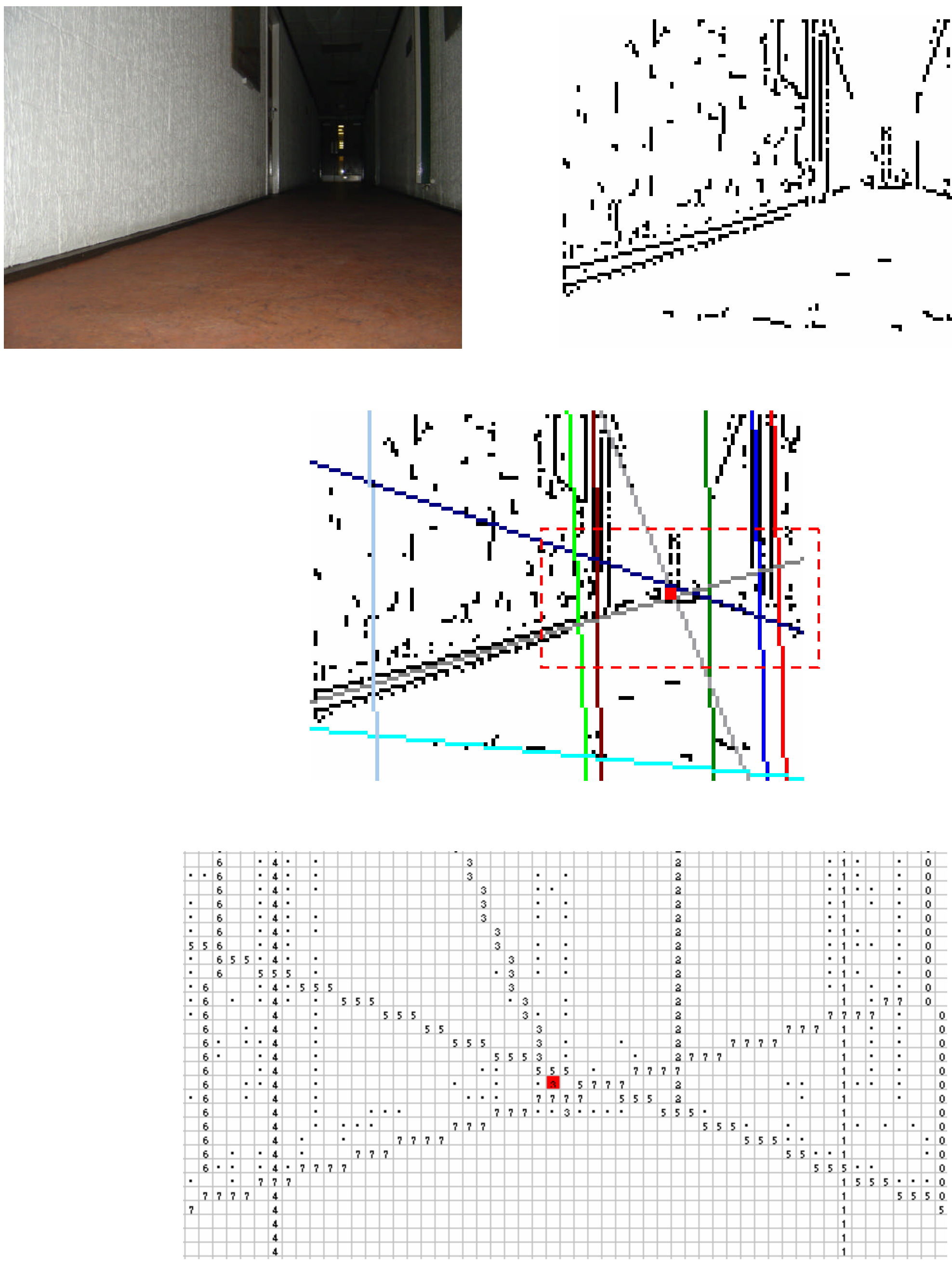


Figure 5.27 Sample result of vanishing point determining algorithm
(a) A typical image (b) Pre-processed version of figure 5.26a image (c) Lines found from figure 5.26a image (d) Area of figure 5.24c image enclosed in red dashed rectangle in numeric format

θ and ρ values for the lines found are shown in table 5.8.

Line id	θ	ρ
0	2	57
1	2	52
2	1	39
3	22	28
4	1	10
5	71	12
6	3	5
7	106	-8
8	84	-40
9	1	-48

Table 5.8 θ and ρ values for lines found in a typical image

The points representing lines 0, 2, 3, 4 and 6 are shown circled and labelled on an extract of the accumulator array in figure 5.28 as well as part of the curve representing the vanishing point (29, 1) found.

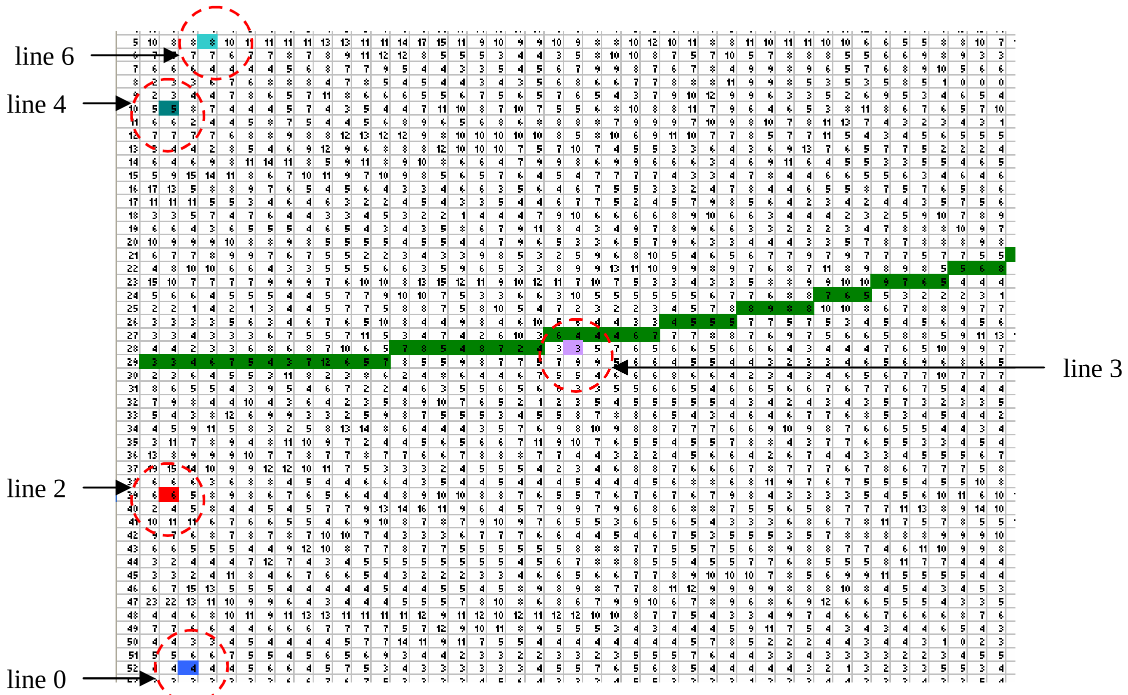


Figure 5.28 An extract of accumulator array showing some points found relative to the curve of accumulator array

The column on the extreme left shows ρ values increasing from 5 to 52. Other columns correspond to the first 44 values of θ. In this figure any point lower than the curve has more positive ρ than the curve and so is positive relative to the curve. In our line classification scheme, the sign of such a line would be positive.

Mathematically, determination of the sign involves two steps:

1. determine the value of ρ on the curve for the vanishing point, ρ_{curve}, that corresponds to the value of θ, θ_{line} for the line under consideration
2. compare ρ_{curve} found from 1 with the value of ρ for the line under consideration, ρ_{line} and if this is less than ρ_{curve}, the sign is negative otherwise it is positive

Step 1 involves calculating

$$\rho_{curve} = x_{VP}\cos(\theta_{line}) + y_{VP}\sin(\theta_{line}) \quad . \quad . \quad . \qquad \text{Equation 5.13}$$

where x_{VP} and y_{VP} are coordinates of the vanishing point, θ_{line} is θ for the line under consideration and ρ_{curve} is ρ on the curve at the point $\theta_{curve} = \theta_{line}$.

Step 2 will then involve the decision

if $(\rho_{line} < \rho_{curve})$ then assign a negative sign to the line's distance
else assign a positive sign to the line's distance

5.3 High Level Features Determination

5.3.1 Corridor(s) Recognition

Corridor recognition is critical to successful navigation in a corridor. In this work, this is achieved by searching for a left-corridor-edge and a right-corridor-edge. These refer to the intersection line between the wall of the corridor on the left and the floor, and the intersection line between the wall of the corridor on the right and the floor, respectively. A simple scheme was devised to achieve this.

To select a left-corridor-edge, points are assigned to lines according to whether or not they fit a certain criterion, and if so, how they score against three further criteria. The line which scores the highest is the left-corridor-edge provided its score is equal to or higher than the cut off score of 11. The criteria are as summarised in table 5.7 and are explained further below:

1. θ value in the range 90° to 180°: This is a necessary criterion (Djekoune and Achour 2000) and lines are only assessed against further criteria if they meet this one.
2. SW to bottom-left-of-image distance: All lines which meet criterion 1 are ranked according to the distance of their most south-westerly point to the bottom left corner of the image. The one with the shortest distance receives a score of 5. The one with the next shortest distance receives 4, etc. Only the lines with the 5 shortest distances receive scores from this criterion.
3. Distance to the VP: All lines which meet criterion 1 are ranked according to their distance to the vanishing point if it has been found. The one with the shortest distance receives a score of 5. The one with the second shortest distance receives 4, etc. Only the lines with the 5 shortest distances receive scores from this criterion.

   If the vanishing point has not been found, no points are scored from this criterion.

4. Line Category: Lines are ranked according to their category as defined in table 5.7. Lines in the horizontal slash and slash categories earn 7 points. Lines in the horizontal category earn 5 points. Lines in the vertical slash category earn 3 points. Lines in the vertical category earn 2 points.

Table 5.9 summarises the criteria.

	Criteria	**Maximum Points**
1	In the range 90 to 180	Necessary
2	SW to bottom-left-of-image distance	5
3	Distance from vanishing point	5
4	Line category	7

Table 5.9 Criteria for selection of left corridor edge

Similarly, the criteria for selection of a right-corridor-edge is summarised in table 5.10:

	Criteria	**Maximum Points**
1	In the range 0 to 90	Necessary
2	SE to bottom-left-of-image distance	5
3	Distance from vanishing point	5
4	Line category	7

Table 5.10 Criteria for selection of right corridor edge

The criteria for selection of a right-corridor-edge differ from those for a left-corridor-edge in a few ways. In criterion 1 for selection of a right-corridor-edge, the range angles of lines must fall in, is 0 to 90. In criterion 2, the distance considered is that of the most south-westerly point on a sub-line to the bottom right of corner of the image. Criterion 3 is exactly the same for both. In criteria 4, lines in the horizontal backslash and backslash categories earn 7 points, lines in the horizontal category earn 5 points, lines in the vertical backslash category earn 3 points and lines in the vertical category earn 2 points.

Sample results are shown in figure 5.29. Sub-lines found as corridor edges are shown in red.

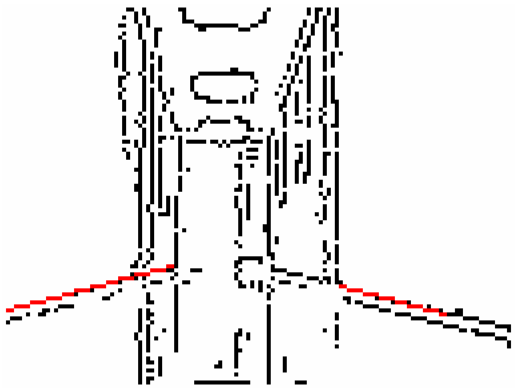


Figure 5.29 Sample result of corridor detection scheme

5.3.2 Door(s) Recognition

With the vanishing point, and line recognition and categorisation algorithms run, the following algorithms can be implemented to detect doors. Broadly, a door can be a corridor-door or a wall-door. A wall-door can be on the left, a wall-door-left, or on the right, a wall-door-right.

5.3.2.1 Corridor-Door Recognition

A corridor door is found by searching for the left, top and right edges of the door.

The left edge is found by checking sub-lines against 2 necessary criteria, and where they meet these criteria, scoring them against 2 further criteria. The necessary criteria are:

1. it's parent line must belong to the vertical category as defined in *5.2.4.1 Line Category*
2. it must be on the left side of the vanishing point, determined from the sign of its parent line as determined in *5.2.4.3 Sign of Displacement of Lines from Vanishing Point*.

The two further criteria are – the distances of their parent lines from the vanishing point, and the difference between the y–coordinate of their midpoint and the y–coordinate of the vanishing point.

The first criterion was introduced because from observing several images, the door edge is usually on one of the first vertical lines encountered scanning from the vanishing point leftwards. The distance is as discussed in *5.2.4.2 Magnitude of Distance for Line from Vanishing Point*. Sub-lines on the closest line score 5 points, sub-lines on the next closest line score 4 points, etc.

The second criterion is necessary to improve scores for sub-lines which are at the right vertical level in the image to be door edges. The sub-line with the smallest difference scores 5, the one with the second smallest distance scores 4, etc.

The right and top edges of the door are determined in a similar manner with minor adjustments. For the right edge, only sub-lines to the right of the vanishing point are considered. For the top edge, horizontal sub-lines above the vanishing point are considered.

The bottom edge of the door will usually not result in a line except when the door is closed.

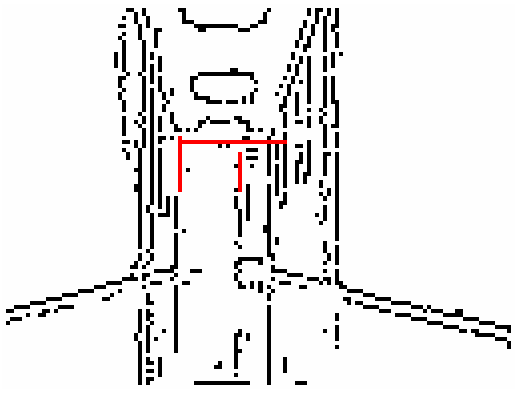


Figure 5.30 Sample door sub-lines found

5.3.2.2 Wall-Door Recognition

To determine that there is a wall-door-right, a slash, or a vertical slash line is sought that does not coincide with the ceiling-wall intersection on the right. To do that, first the slash (or vertical slash or horizontal slash) sub-line corresponding to the ceiling-wall intersection is found, and then all other slash and vertical slash lines just below it are found. They are checked to see if any of them has a sub-line which has two vertical lines approximately at its ends which satisfy:

1. have a sub-line whose top edge approximately intersects with the top edge sub-line under consideration
2. have a sub-line which approximately intersects with the right-corridor-edge

Figure 5.31 shows a sample result. The top of the door is drawn in green. The two sub-lines of both sides which are edges of the door found are shown in red and blue for the first and second sub-lines found respectively.

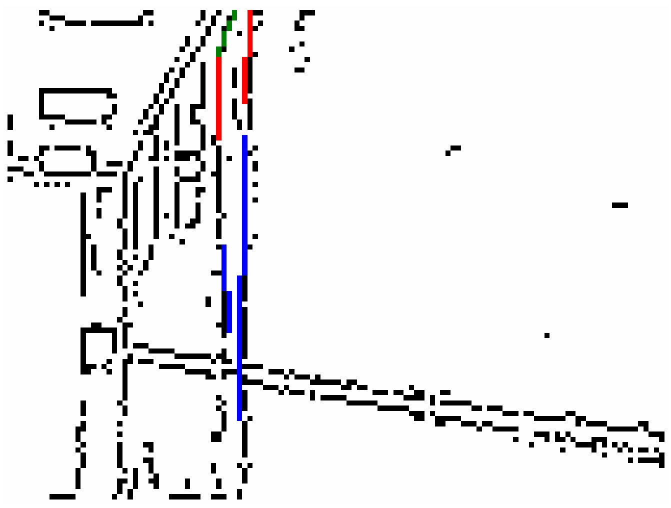


Figure 5.31 Sample wall-door-right detected

The nature of the images encountered is such that the slash line corresponding to the top of a wall-door is not easily detected, so if a wall-door is not found, the image is checked for vertical sub-lines extending from the top of the wall where the door is to be found, to the bottom of it. Where such vertical sub-lines are found, it is possible that a wall door exists on the wall, and another attempt should be made to find it again after the robot has moved a little.

A wall-door-left is found in a similar way except that a backslash and vertical backslash are the target categories of lines rather than slash and vertical slash respectively, and other similar adjustments are made.

5.4 Navigation

Actual navigation is designed to depend on the objective of the robot. Three simple navigation objectives have been defined – move-along-corridor, navigate-into-door-on-the-left, and navigate-into-door-on-the-right. These simple objectives can be combined in any way to derive more sophisticated and more useful objectives.

5.4.1 Move-Along-Corridor

To move along the corridor, the robot first tries to locate the vanishing point. If it is found, then if it is around the centre of the image, the robot can move straight along. If the vanishing point is not approximately at the centre, then the robot needs to move in such a way that the vanishing point moves towards the centre of the image. This means, for example, that if it is slightly to the left, the robot needs to move slightly to the left, and if the vanishing point is to the far right of the image, the robot needs to turn right considerably.

If the vanishing point was not found, it checks to see if a left-corridor-edge or a right-corridor-edge was found. If a left-corridor-edge is found, the robot turns slight to the right, and if a right-corridor-edge is found, it turns slightly to the left.

If no vanishing point or corridor edge is found, the robot turns at a spot, i.e. while stationary through an angle of 60° and tries again.

5.4.2 Turn into Wall-Door

Coming down a corridor with the intention of turning into a wall-door-right, the robot needs to look out for at least 5 stages.

1. In the first stage, the door, if it exists is so far away that its top edge is so small it cannot be detected as a valid sub-image. It's left and right edges are identifiable as two vertical lines running from the top of the corridor to the bottom of the corridor.
2. In the second stage, the robot is near enough to the door to detect the top edge of the wall-door-right and so make a positive identification of the door using the procedure described in *5.3.2.2 Wall-Door Recognition*.
3. In the third stage, the robot has moved so close to the door it can no longer see the top edge as it is beyond the image. It can however see two vertical lines which by now would be very clear, running from the top of the image to the floor of the corridor.
4. After a while, the side of the door nearest to it is no longer visible as it has gone out of the field of view of its camera. It can however see the other edge.

When stage 4 is attained, the robot needs to start turning gently to the right while still moving until it looses the far edge of the door while looking out for the near edge of the door which it lost in stage 4. If it sees the near edge, it has overturned and needs to turn left slightly while looking out for the far edge etc.

A similar process can be used to turn into a wall-door-left with appropriate minor adjustments.

Chapter 6 Mobile Robot Vision Systems based on Line Detection using Artificial Neural Networks

Various approaches can be taken to achieve vision for robot navigation using artificial neural networks. A few of them are discussed in detail in *2.4 Artificial Neural Networks in Vision Systems for Mobile Robot Navigation*. Two new approaches were investigated in some detail in this work, both with the aim of mimicking the Hough transform in the sense that both of them attempt to achieve vision for navigation by recognition of straight lines.

One approach attempted to find lines belonging to the categories set out in table 5.7. It would look for lines in each category with a neural network trained to recognise lines in that category in parallelograms extracted from the image which cover the area in the image where lines with all possible values of θ for the given category cover, for a given distance from the origin. The network would say whether a line in the target category exists in the input parallelogram. The first possible parallelogram would first be extracted and passed to the network, and then the parallelogram would be slided along the width of the image at a pre-chosen interval. Lines found for all parallelograms can then be further analysed for existence of high level features. This in a way imitates standard line-detection with the Hough transform as described in Chapter 5. This approach did not do very well in recognition of lines except for the vertical category for which target lines tended to be long. The implementation and results of this approach are discussed further in *6.2 Robot Vision System based on Full Line Detect.*

The second approach involved breaking an image down to sub-images, and then determining whether the sub-image contains a line in any of the categories defined in table 5.7 using a group of 8 neural networks called the stage 1 networks. Results from these stage 1 networks are then passed to a stage 2 network which is set up to work out the direction the robot should move in. This in a way imitates vision using what is called the hierarchical Hough transform. The stage 1 networks did reasonable well in recognising the target lines, but the stage 2 network did not do well in determining the direction in which the robot should move. This approach is discussed in more detail in *6.3 Vision for Navigation based on Sub-Images Processing.*

Although both approaches did not do particularly well by themselves, the second approach was useful, along with the Hough transform in a hybrid approach discussed in *7.2.2 Hybrid 5: Use ANNs to find lines in Sub-Images and then use Hough transform to establish 'full-picture'*.

6.1 Training Parameters for Back-Propagation Networks Used

2.2.4 The Back Propagation Network introduced the back propagation network, and how it has been used in this work. This section carries this on by specifying some of the parameters employed for this work. Most of the implementation in this work is based on (Rogers 1996), and further details can be found there, as well as countless other materials on the subject.

In summary, the training method used is summarised as follows:

```
backPropTraining
{
        initialise iterationCount to 0
        while numOfTrainedPatterns < NumInTraining
        {
                Initialise numOfTrainedPatterns to 0
                forall patternsInTraining, p
                {
                        place p on the network
                        do forward pass
                        determine error for p
                        do backward pass
                        if error for p is less than threshold
                                increase numOfTrainedPatterns by 1
                }
                increase iterationCount by 1

                if iterationCount == maxNumOfIterationsAllowed
                        break
        }

if iterationCount < maxNumOfIterationsAllowed
        save network parameters
else
        declare that training failed
}//end backPropTraining
```

Inputs to the process include a training set, a network set up with random weights for its links.

At the heart of the process are two loops, one nested in the other.

6.1.1 Outer Loop

The outer loop, shown above as a while loop, runs until every pattern p, in the training set conforms to the training criteria, i.e., yields an error when passed through the current network, which is less than a pre-defined threshold. In other words, the outer loop runs until the number of patterns that have conformed, or have been trained, numOfTrainedPatterns, equals the total number of patterns NumInTraining. The loop also increments iterationCount by 1 each time it is run, and monitors it so it does not go beyond a predetermined threshold, maxNumOfIterationsAllowed. iterationCount is initialised to 0 before the outer loop starts, and if it does get to maxNumOfIterationsAllowed, the training process is halted and training is judged to have failed.

Various networks will be considered in the rest of this chapter, and more specific information about parameters such as the maximum number of iterations allowed, and where training is deemed to have failed, how this decision was reached, will be discussed at the point the specific network is being considered.

6.1.2 Inner Loop

The inner loop passes individual patterns forward through the network, determines whether or not the error from the pass is lower than the error threshold, and update the count of trained patterns, numOfTrainedPatterns. It also does a back pass which adjusts the weights of the network to 'fit in' the current pattern better.

The forward pass, as mentioned in *2.2.4 The Back Propagation Network,* uses the sigmoid function to assign values to nodes. This function is shown:

$$y = \frac{1}{1+e^{-NET}} \qquad . \qquad . \qquad . \qquad \text{Equation 6.1}$$

where NET for a particular node is the sum of the product of the weight of all links coming into that node, and the value of the node that the particular link originates from. Value is determined for all nodes except for those in the input layer.

Errors are determined for output nodes by subtracting the actual outputs from the node from the target outputs, and for the pattern by summing the absolute value of all the errors of its output nodes. Patterns with errors exceeding a predefined threshold are counted using the variable numOfTrainedPatterns, as pointed out earlier in *6.1.1 Outer Loop*.

Perhaps the most defining step of the training in the back-propagation method is the back pass. It involves propaging the error for the pattern back through the network by adjusting the weights on its links using what is known as the delta rule. By this rule, an adjustment, Δ_i, is worked out for each link i using

$$\Delta_i = \eta x_i \delta \quad . \quad . \quad . \qquad \text{Equation 6.2}$$

where

$$\delta_i = y_i(1-y_i)(d_i - y_i) \quad . \quad . \quad . \qquad \text{Equation 6.3}$$

for the output neurons, and

$$\delta_p(q) = x_p(q)[1-x_p(q)]\sum w_{p+1}(q,i)\delta_{p+1}(i) \quad . \quad . \quad . \qquad \text{Equation 6.4}$$

for neuron q in hidden layer p.

η in equation 6.2 is the learning rate, introduced in *2.2.4 The Back Propagation Network*. y_i and d_i are actual and desired outputs respectively, in equation 6.3. For Hidden layer neurons, $w_{p+1}(q,i)$ is the weight of the link ending in layer $p+1$ (the next layer from the current one p), starting from node q in layer p and ending in node i (which is in layer $p+1$).

6.2 Robot Vision System based on Full Line Detection

This approach attempts to imitate line-detection as done in the standard straight line Hough transform. Networks were designed to detect lines from each of the eight categories detected with the Hough transform (vertical, horizontal, vertical slash, slash, horizontal slash, vertical backslash, backslash, horizontal backslash). Characteristics of the categories are summarised in table 5.7.

Four steps were involved for each line category:

1. Determine the dimensions for the parallelogram for the category in pixels.
2. Develop a training set for that category with lines found from actual images by extracting parallelograms of the right size from the training images.

3. Set up and train the network
4. Test the network

6.2.1 Vertical Category Lines Recognition with Full Line Detection

6.2.1.1 Width of Vertical Category

To determine the parallelogram for the vertical category, two lines were considered from the same value of ρ at 0 – one was drawn with the minimum value of θ in this category, and the other with the maximum value. From table 5.7, the minimum value of θ in this category is -5° (or 175°), and the maximum value is 4°. The two lines resulting are shown in figure 6.1.

By working out the endpoints of lines AC and BD, the vertices of the parallelogram $ABCD$ formed can be determined. The top left vertex, A, is at point (-3,47), top right, B, is at point (4,47), bottom left, D, is at (-4,-48) and bottom right, C, is at (3,-48).

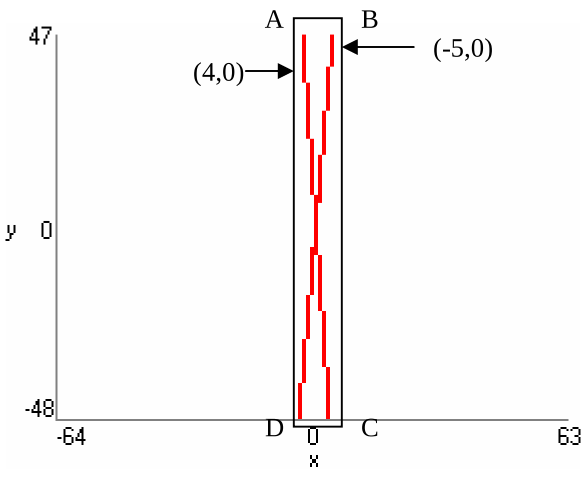


Figure 6.1 Width of parallelogram containing full range of vertical category lines at ρ = 0

The width of parallelogram $ABCD$ which is the horizontal distance between line AD and line BC inclusive is 4-(-4) + 1 or 9 pixels. The height of the stripe is the height of the image, 96 pixels.

6.2.1.2 Training and Testing

Lines found using the Hough transform for actual images were used to develop a training set. They were extracted along with the 9 x 96-sized parallelogram around them. For example, if a vertical line in an actual image has a range from $x = 14$ to $x = 17$, it occupies a 4 column width, and so needs to be padded to 9 columns.

The parallelogram ranging from 12 to 20 is included in the training set. Outputs corresponding to parallelograms with vertical lines were set to 1, and outputs corresponding to parallelograms that did not contain vertical lines were set to 0.

56 9 x 96 parallelograms from several images constituted the training set.

A 4 layer back-propagation neural network was set up with 9 x 96, i.e., 864 inputs and 1 output. The first inner layer had 432 neurons and the second inner layer had 216 neurons.

When tested with a random selection of 34 stripes from actual images, 10 of which should have vertical lines, 7 were recognised correctly as having vertical lines and 3 which should have vertical lines were reported as not having them. 3 of the stripes which should not have vertical lines were returned as having vertical lines. 21 stripes which should not have vertical lines were correctly recognised as not having them.

Although this result is not very good in back-propagation network terms, it is not altogether bad, and for a system such as the mobile robot system under consideration, errors can be tolerated for some lines in some images as this will not necessary completely derail the robot. On the whole, it is passible.

6.2.2 Vertical Backslash Category Lines Recognition

6.2.2.1 Width of Category

The minimum value of θ for the vertical backslash category is 5°, and maximum value is 24°.

The two extreme lines in this category where $\rho = 0$, are shown in figure 6.2. The area a line in this category must fall into, is shown as the parallelogram $ABCD$. Top left vertex, A, is at point (-21,47), top right vertex, B, is at point (-4,47), bottom left, D, is at (4,-48) and bottom right, C, is at (21,-48).

An image can be divided into stripes of the required type by sliding the parallelogram from the point when B on the parallelogram coincides with (-64, 47). The starting parallelogram is $A(-81,47)$, $B(-64,47)$, $C(-39,-48)$ and $D(-56,-48)$.

Every line in this category will lie in a parallelogram with width -4-(-21) + 1 or 18.

To extract the points in the parallelogram, on the first row, the point coinciding with vertex A on the parallelogram, and the 17 points following are taken. For other rows, points on the row of the image are scanned until (x,y) is on or just beyond line AD. The next 17 points are then added to the parallelogram.

To determine whether or not (x,y) is on or just beyond AD, y in (x,y) is compared to y', calculated from

$$y' = mx + c \qquad . \qquad . \qquad . \qquad \text{Equation 6.5}$$

where $m = -3.8$ from (-64-(-39))/(47-(-48)) considering points B and C.

The intercept c_k for successive ADs in subsequent parallelograms is

$$c_k = y_{\max} - mx_k \qquad . \qquad . \qquad . \qquad \text{Equation 6.6}$$

where $y_{\max} = 47$ is y for the top row and x_k is x for the k^{th} point under consideration.

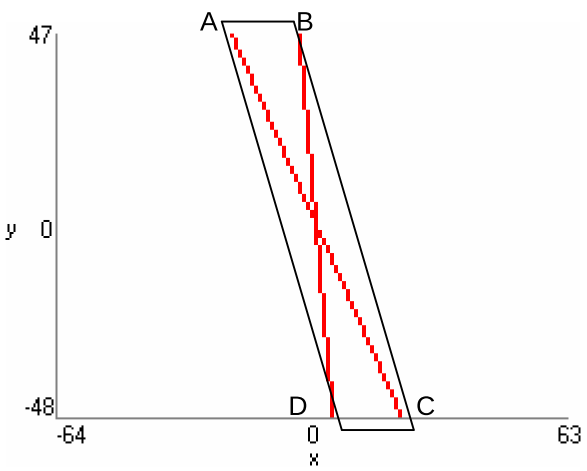


Figure 6.2 Parallelogram containing full range of vertical backslash category lines

Beginning from -64, for a point in any row to lie in this parallelogram, it must lie between line AD defined by $y = -3.8x - 32.8$ and line BC defined by $y = -3.8x + 31.8$.

6.2.2.2 Training and Testing

Lines found using the Hough transform for actual images were used to develop a training set. 78 18 x 96 parallelograms from images made up the training set but no significant success was achieved when the network was tested with fresh images.

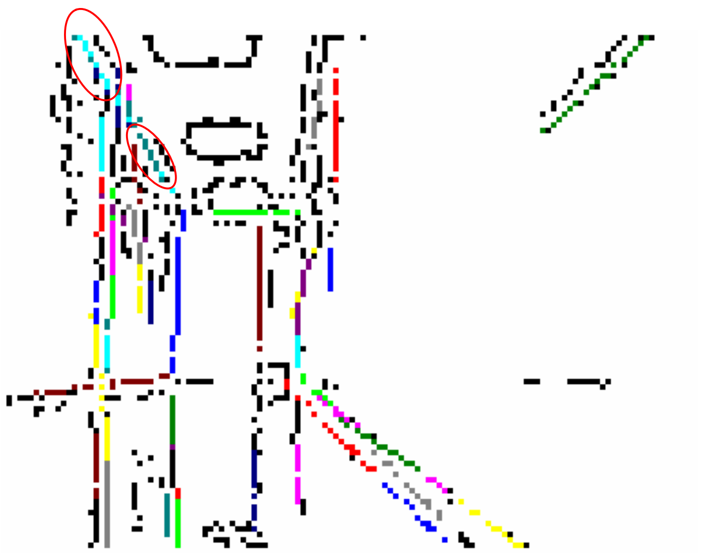


Figure 6.3 Some vertical backslash lines not found by full line detecting neural network for the vertical backslash category

This is likely to be because sub-lines in this category tended to be very short, and so were not significant parts of the training and testing stripes. The image shown in figure 6.3 exemplifies this. The lines that should have been detected are shown with a red ellipse around them. None of them was found when this particular image was used for testing.

Various 3 and 4 layer architectures were tried for the network.

6.2.3 Conclusion

Because results from *6.2.2 Vertical Backslash Category Lines Recognition* were so poor, it was resolved this approach of imitating the standard straight line Hough transform is not very promising. It worked quite well for a category like the vertical category in which lines tend to be very long. It failed woefully for a slanted category for which lines tend to be very short relative to the stripes that contain them. The approach was aborted.

6.3 *Vision for Navigation based on Sub-Images Processing*

Using this approach, the thinned 128 x 96 sized image obtained from pre-processing (discussed in *Chapter 3 Pre-processing*) is broken down to 8 x 8 sized sub-images. Figure 6.4 illustrates this.

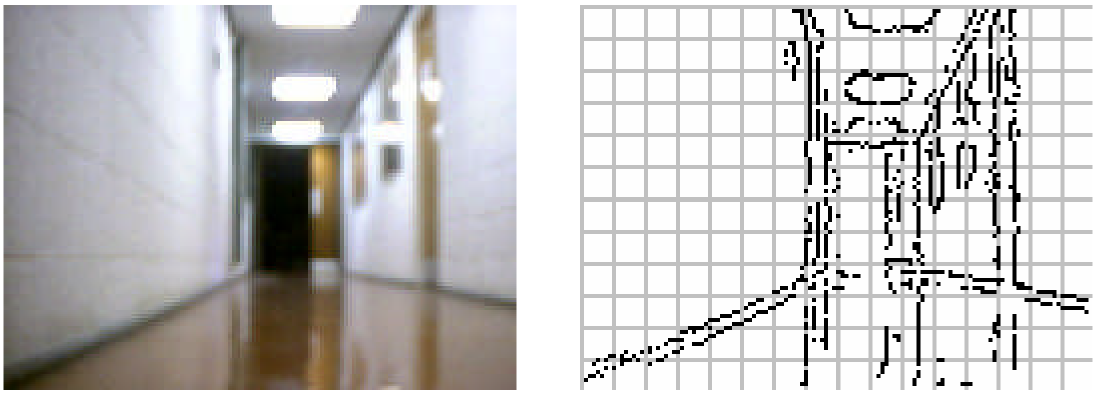


Figure 6.4 Pre-processed image broken down into 8x8 sized sub-images

Sub-images are labelled with identification codes illustrated in Table 6.1. The sub-image at the top-left position is labelled 0. Subsequent sub-images going right are labelled with consecutive numbers until the end of the row. The labelling is continued on the next row from the left.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191

Table 6.1 Sub-image labelling order

The 64 pixels in the sub-images constituted input to the neural networks used. 8 separate neural networks are trained to determine if the sub-images contain lines from each of the 8 categories detailed in table 5.7. Output from each of these networks is a binary digit which indicates whether the sub-image contains a line of the category the network is trained to detect. These networks are described further in *6.3.1 Line Recognition and Categorisation from Sub-Images.*

A second network then aims at putting together the findings of these line recognition and categorisation networks, and from them determining the direction the robot should take. This is discussed further in *6.3.2 Processing 'full picture'.*

6.3.1 Recognition of Lines in Categories from Sub-Images

8 networks were set up to recognise lines in each of the categories described earlier, from 8 x 8 sized sub-images extracted by breaking the image down. The networks therefore have 64 binary inputs. Each network has a single output which has a value of 1 if a line which falls into the corresponding category is detected in the input sub-image and 0 otherwise. The networks also have 1 inner layer with 9 neurons.

A training set was developed incrementally from sub-images taken from 5 randomly selected images. Training was performed, and the network was tested with a fresh random image. Sub-images which are not correctly identified are added to the training set, and the network is re-trained. This was done until further additions to the training set did not significantly improve the recognition rate in fresh random images. 216 sub-images were derived in this way.

In the training for each network, further sub-images were included which contain clear instances of lines in the category. This was necessary because certain categories do not occur commonly in actual images so there were not enough of them to properly train the networks. The final training set contained 226 to 263 sub-images for the various categories. The target outputs were adjusted appropriately for each sub-image in the training set, when training for each category.

Final testing was performed for each category with at least all the 192 sub-images from a complete image.

The sub-sections which follow discuss the training and testing for each line category in more detail.

6.3.1.1 Vertical Category Lines Recognition

Lines in this category have θ values in the range -5° to 4°. How lines within this category appear in sub-images is illustrated in the figure 6.5. For the purpose of this illustration, they are both drawn to pass through the centre of the image with $\rho = 0$. The figure shows various possible appearances of the lines in sub-images, and was used as a guide during training.

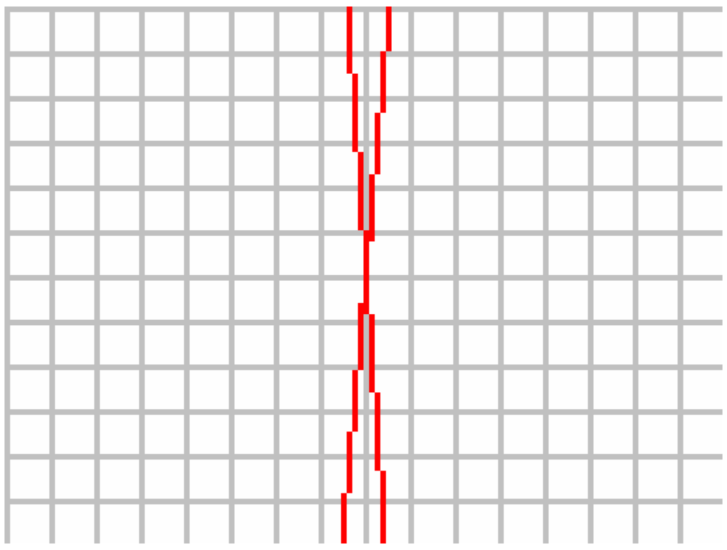


Figure 6.5 Possible appearances of extreme vertical category lines in sub-images

The network for this category failed to correctly classify 5, 12, 14 and 15 of the 192 sub-images in 4 random test images.

The final training set is contained in *appendix C* along with details of the test for one the random images. Test results for another random image which is shown in figure 6.6 are illustrated in figure 6.7.

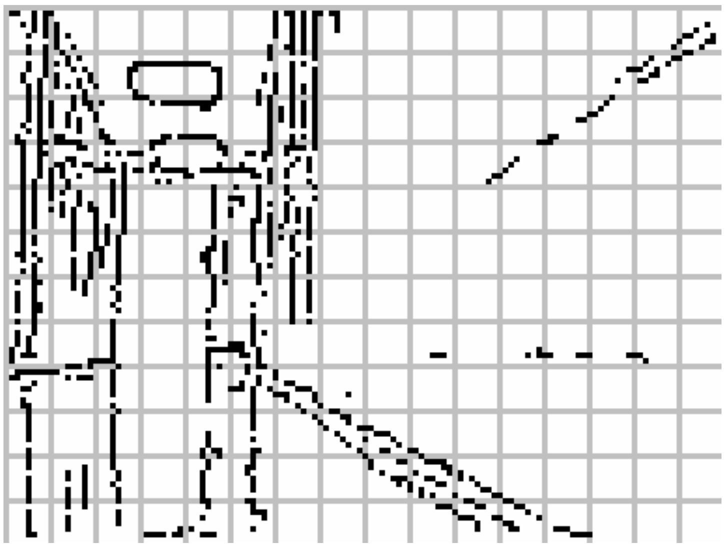


Figure 6.6 Broken down image used for testing

Red lines in a sub-image indicate that a vertical line was found in the sub-image. Note that the red lines are drawn in the middle of the sub-image and do not indicate the actual position within the sub-image where the line was found. Information about the actual position or arrangement of pixels in the line, or whether more than one vertical line exists, is not obtained with this approach.

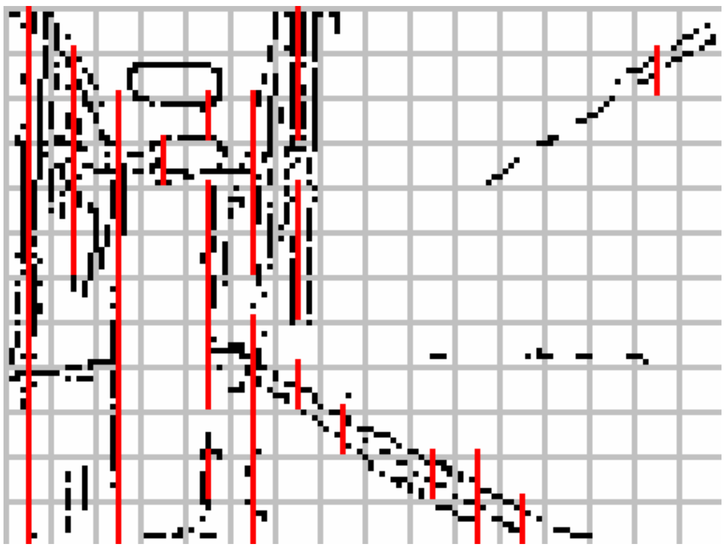


Figure 6.7 Results of test for vertical category

6.3.1.2 Vertical Backslash Category Lines Recognition

Lines in this category have θ values in the range 5° to 24°. How lines with these extreme θ values for this category appear in sub-images is illustrated in the figure 6.6. For the purpose of this illustration, they are both drawn to pass through the centre of the image where $\rho = 0$.

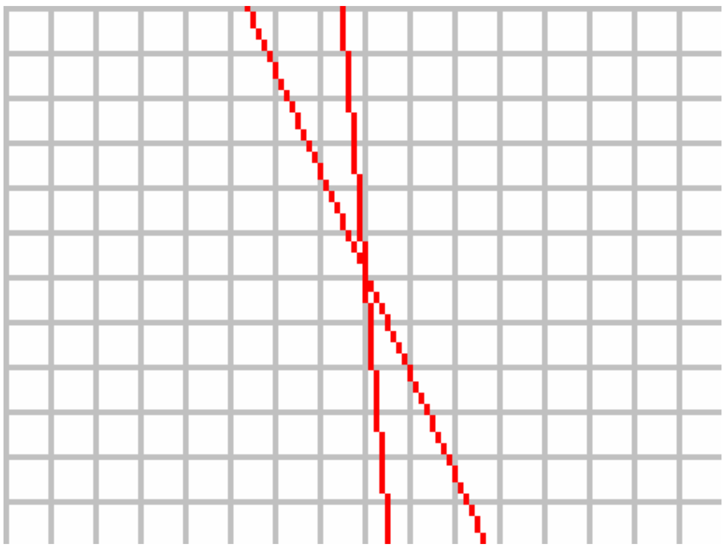


Figure 6.8 Possible appearances of extreme vertical backslash category lines in sub-images

Note that some lines in this category appear like vertical lines in within some sub-images.

Testing was done with the same random image used for the test in *6.3.1.2 Vertical Backslash Category Lines Recognition for* which results are stored in *appendix C* and the 14 misrecognitions were found. Test results for the random image which is shown in figure 6.6 are illustrated in figure 6.9.

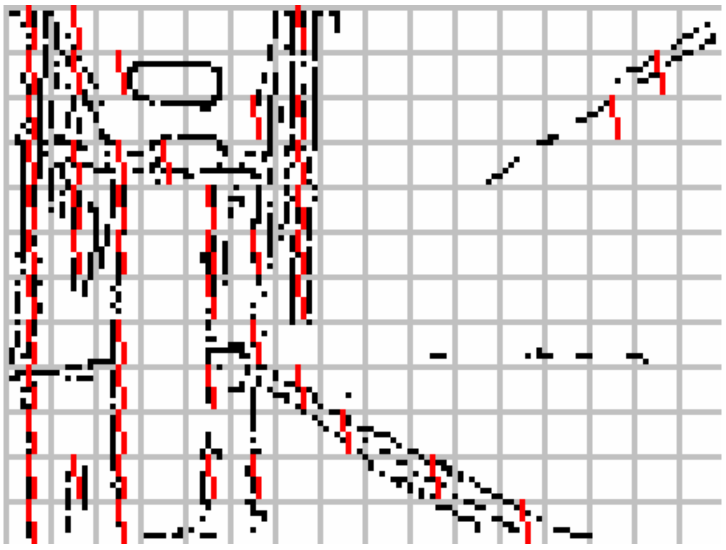


Figure 6.9 Results of test for vertical backslash category

6.3.1.3 Backslash Category Lines Recognition

Lines in this category have θ values in the range 25° to 64°. How lines with these extreme θ values for this category appear in sub-images is illustrated in figure

6.10. For the purpose of this illustration, they are both drawn to pass through the centre of the image with $\rho = 0$.

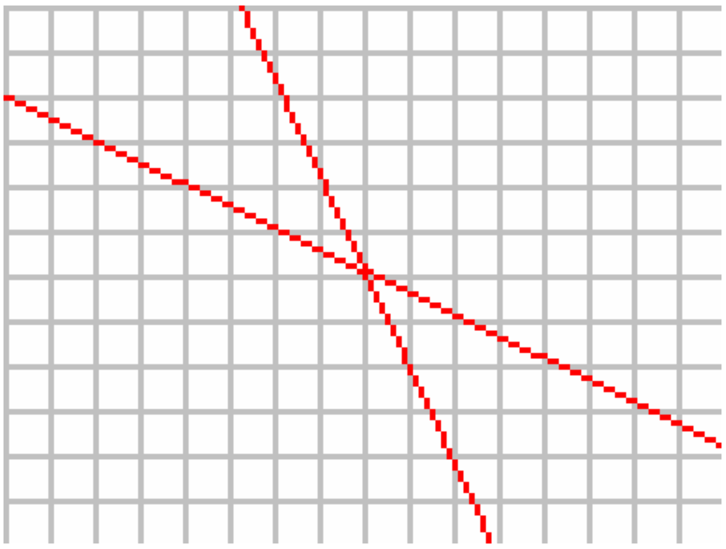


Figure 6.10 Possible appearances of extreme backslash category lines in sub-images

The network was tested with the first random image from *6.3.1.2 Vertical Backslash Category Lines Recognition* and 11 sub-images were misrecognised. Test results for the image shown in figure 6.6, are illustrated in figure 6.11.

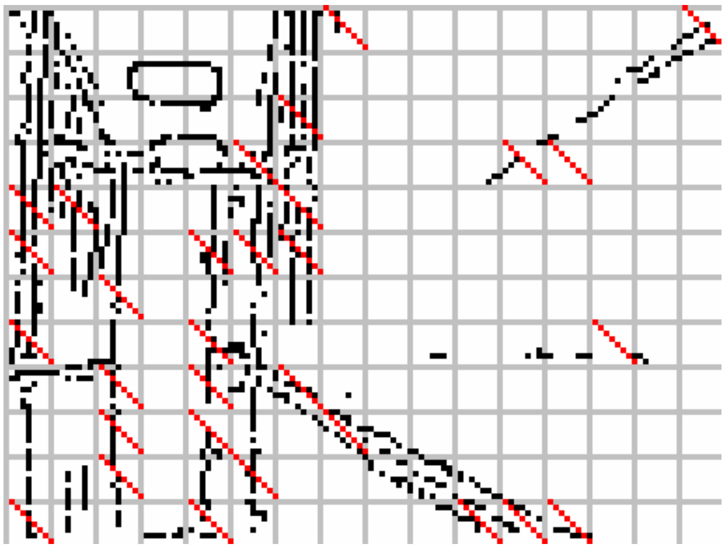


Figure 6.11 Results of test for backslash category

6.3.1.4 Horizontal Backslash Category Lines Recognition

Lines in this category have θ values in the range 65° to 84°. How lines with these extreme θ values for this category appear in sub-images is illustrated in figure 6.12. For the purpose of this illustration, they are both drawn to pass through the centre of the image with $\rho = 0$.

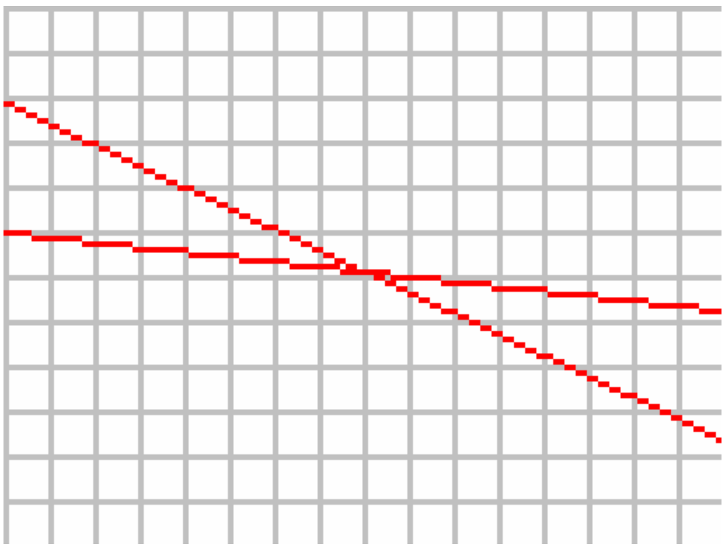


Figure 6.12 Possible appearances of extreme horizontal backslash category lines in sub-images

When tested with the usual first random image, 14 sub-images were wrongly identified as horizontal backslashes. Most of the 14 were very similar to backslashes so the “mistakes” are ‘understandable’. 4 horizontal backslashes were not picked up by the network. 20 horizontal backslashes were correctly identified, and all other sub-images were correctly identified as not being horizontal backslashes. Test results for the usual random image, the one shown in figure 6.6, are illustrated in figure 6.13.

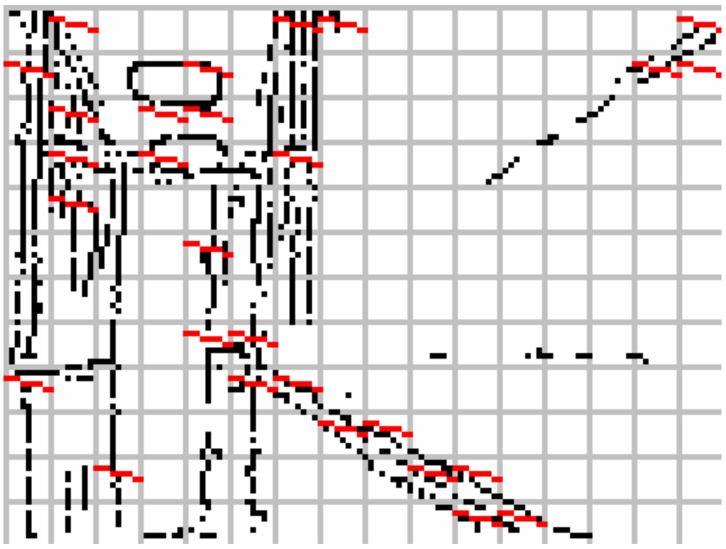


Figure 6.13 Results of test for horizontal backslash category

6.3.1.5 Horizontal Category Lines Recognition

Lines in this category have θ values in the range 85° to 94°. How lines with these extreme θ values for this category appear in sub-images is illustrated in figure

6.14. For the purpose of this illustration, they are both drawn to pass through the centre of the image with $\rho = 0$.

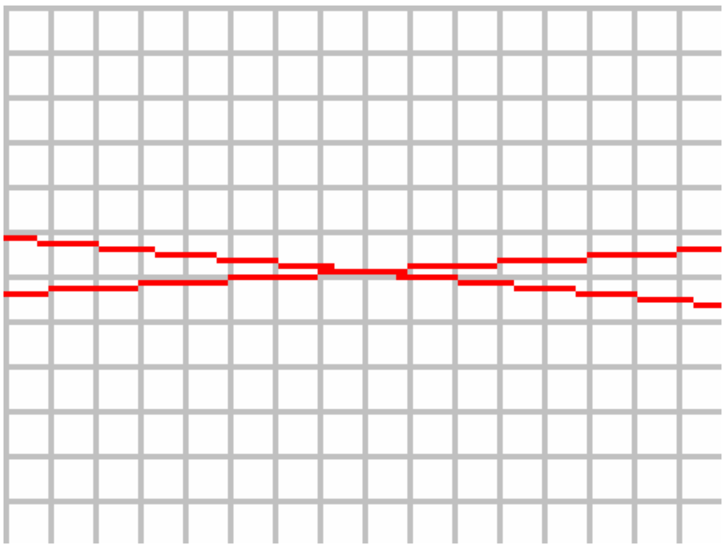


Figure 6.14 Possible appearances of extreme horizontal category lines in sub-images

When tested with a random image, 25 sub-images were correctly identified as horizontal lines, 23 were wrongly classified as horizontal lines, 10 of which were "understandable", and 5 horizontal lines were missed. 4 horizontal backslashes were not picked up by the network. 140 sub-images were correctly identified as not being horizontal backslashes. Test results for the random image shown in figure 6.6, are illustrated in figure 6.15.

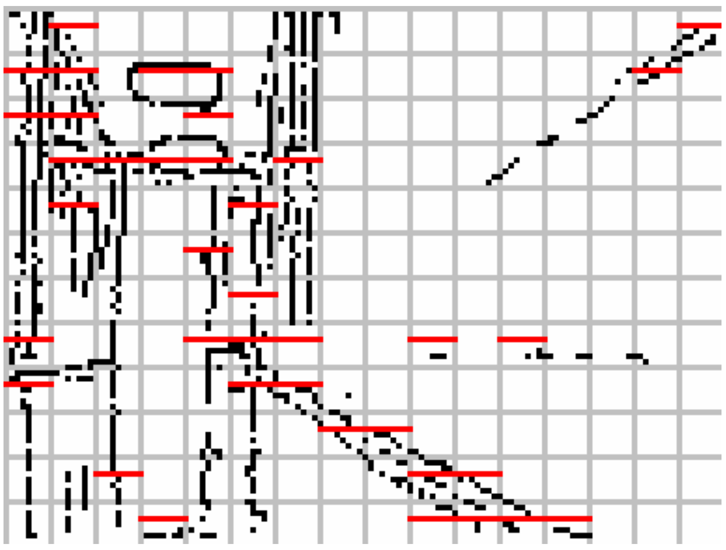


Figure 6.15 Results of test for backslash category

6.3.1.6 Horizontal Slash Category Lines Recognition

Lines in this category have θ values in the range 95° to 114°. Their appearance is similar to the appearance of horizontal backslash lines.

When tested with a random image, 25 were wrong, 25 were recognised correctly, and none was missed. Results for test with the image in figure 6.6 are illustrated in figure 6.16.

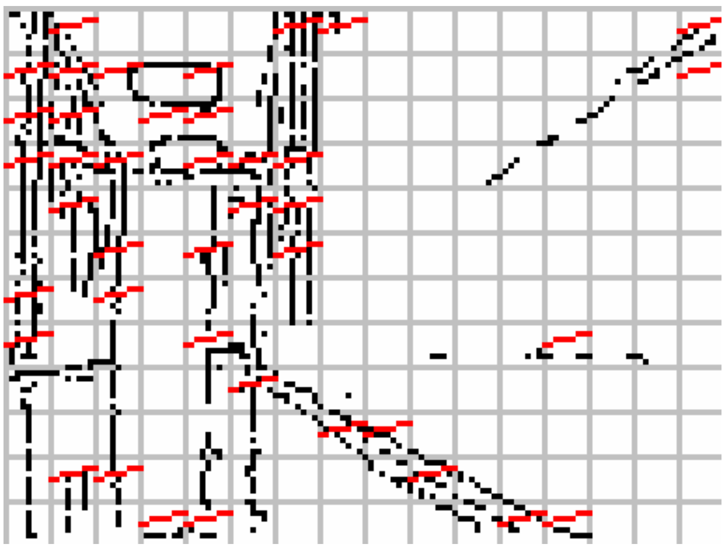


Figure 6.16 Results of test for horizontal slash category

6.3.1.7 Slash Category Lines Recognition

Lines in this category have θ values in the range 115° to 154°. There appearance is similar to the appearance of backslash lines.

When tested with a random image, 9 were wrong, 6 were recognised correctly, and 2 were missed. Results for test with the image in figure 6.6 are illustrated in figure 6.17

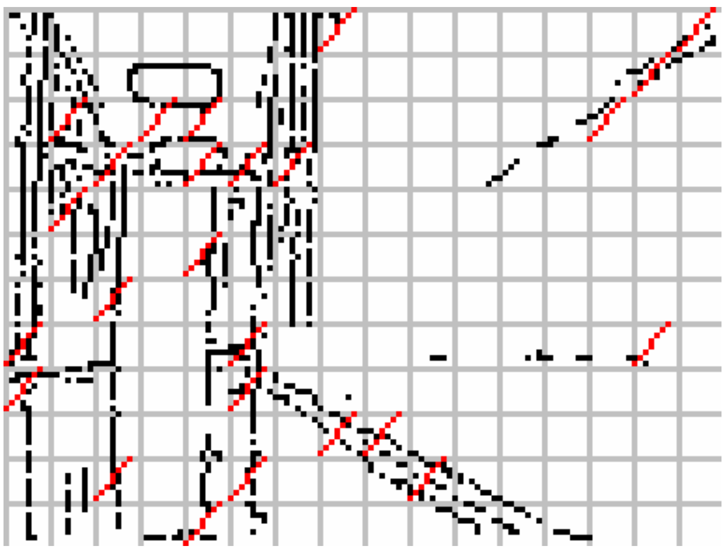


Figure 6.17 Results of test for slash category

6.3.1.8 Vertical Slash Category Lines Recognition

Lines in this category have θ values in the range 155° to 174°. There appearance can be deduced from the appearance of lines in the vertical backslash category.

When tested with a random image, 11 were wrong, 33 were recognised correctly, and 11 were missed. Results for test with the image in figure 6.6 are illustrated in figure 6.18

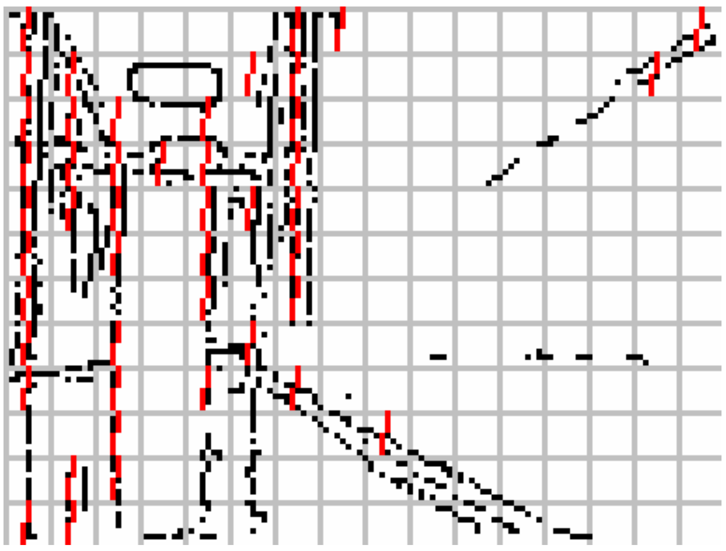


Figure 6.18 Results of test for vertical slash category

6.3.1.9 Results

All the results from the various networks are put together in an 8-tuple vector for each sub-image when the networks have all run. Table 6.2 summarises the configuration for each vector.

Vector Position	Meaning of Possible Value	
	0	1
0	No vertical line found	Vertical line found
1	No vertical backslash line found	Vertical backslash line found
2	No backslash line found	Backslash line found
3	No horizontal backslash line found	Horizontal backslash line found
4	No horizontal line found	Horizontal line found
5	No horizontal slash line found	Horizontal slash line found
6	No slash line found	Slash line found

7	No vertical slash line found	Vertical slash line found

Table 6.2 Configuration of results vector

As an example, the vector 00011100 from a sub-image would mean that for the particular sub-image, a horizontal backslash line, a horizontal line, and a horizontal slash line were found.

An example from an actual sub-image is shown in figure 6.19. Figure 6.19a is a typical sub-image, and figure 6.19b is the results vector obtained when it is processed as described.

```
0 0 0 0 0 0 0 0
0 1 1 0 1 0 0 0
0 1 0 0 1 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0
0 1 0 0 0 0 0 0
0 1 0 0 1 0 0 0
0 0 1 0 1 0 0 0                1 1 0 0 0 0 0 1
```

Figure 6.19 Sample combined result of sub-lines detection in sub-image using ANNs
(a) Sample sub-image (b) Sample result vector for figure 6.19a sub-image

In this example, the result shows that there is a vertical line (the first element of the vector is 1), a vertical backslash line (the second element of the vector is 1) and a vertical slash line (the eight element of the vector is 1). The patterns which suggest these are circled in figure 6.19a.

It is important to remember that although all the lines in the example appear to be vertical lines in the sub-image, they may in fact be parts of a vertical backslash line, or a vertical slash line as suggested by the results vector, and as can be seen by studying figure 6.7.

A full set of result vectors for the image in figure 6.6 are available in *appendix D*. The combined results are also illustrated graphically in figure 6.19 below. Note that some of the coloured lines in the figure overlap.

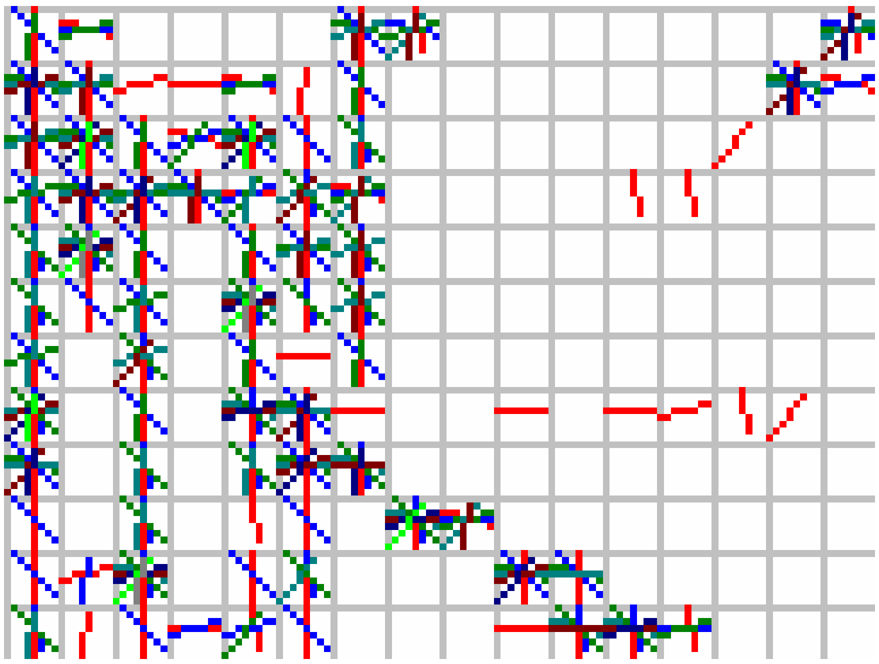


Figure 6.20 Results from line detecting networks for image in figure 6.6

Different colours are used to represent the different lines found in each sub-image. The first line found in each sub-image is coloured red, the second one is blue, the third one is green, etc. Table 6.3 summarises all the colours corresponding to the orders in which the lines are found:

Category ID	Colour	Sample Line
0	red	▬
1	blue	▬
2	green	▬
3	teal	▬
4	maroon	▬
5	navy	▬
6	lime	▬
7	dark gray	▬

Table 6.3 Colours used to indicate different lines found in a sub-image

Again it should be noted that the system developed cannot specify where in the sub-image a line was found, and all lines are shown centred in the sub-image in which they were found.

6.3.2 Processing ‘Full Picture’

A network was set up to try and locate the vanishing point from results of sub-image processes. Its input consisted of the 8 element binary vector of results for each sub-image for each sub-image described in *6.3.1.9 Results* above. Maximum number of training iterations allowed was set at 200,000.

The total number of input nodes is 8 x 192, 192 being the total number of sub-images from each image.

Two forms of output were tried. In one form, there were 128 elements representing pixels across the width of the image. One element was set to 1 to denote the position of the vanishing point and all others were set to 0. In the other form, there were 16 elements representing the 16 sub-images possible across the image. With both forms training could not be achieved with 2 patterns although several network architectures, learning rates and momentum terms were used. It was decided the network was not feasible.

A third approach was tried with only the results from the vertical line recognition being used as input, and the 16 bit output was used. Training was achieved with 2 patterns but not with more and that was not enough to test reasonably. All test results were wrong.

In conclusion, all the approaches tried for putting together lines from sub-images did not yield any tangible results.

Chapter 7 Hybrid Hough Transform/Neural Networks Vision Systems for a Mobile Robot

This chapter discusses vision systems for navigation for a mobile robot that are hybrids of the Hough transform and the artificial neural networks systems discussed in Chapters 5 and 6.

Two groups of possible hybrids are identifiable. One group involve the use of Artificial Neural Networks in what was originally a Hough transform system. The second group involve the use of the Hough transform in a system designed to work with artificial neural networks and artificial neural network paradigms.

7.1 Hybrids with Artificial Neural Networks in Hough Transform System

A few hybrid possibilities were identified in this category. They are discussed in the sub-sections *7.1.1 Hybrid 1: Use of an ANN to Map from Image Space to Peaks in Accumulator Array, 7.1.2 Hybrid 2: Use of an ANN for Peak Detection* and *7.1.3 Hybrid 3: Use of an ANN to Map from Peaks in Binary Accumulator Array to Vanishing Points*.

7.1.1 Hybrid 1: Use of an ANN to Map from Image Space to Peaks in Accumulator Array

This approach attempts to use a neural network to perform the Hough transform algorithm. It takes as input the pre-processed image which would normally be the input for the Hough transform. Its output is a binary accumulator array with peak elements in the accumulator array having a value of 1 and other elements having a value of 0.

The size of the input is the size of the pre-processed image, i.e., 128 x 96 or 12288. The size of the output is the size of the accumulator array 186x160, i.e. 29760. A 100-node middle layer was used.

Training could not be achieved for an initial training set of 2 patterns after 200,000 attempts with various network topologies and values for learning rate and momentum term. No further progress could therefore be made with this approach.

7.1.2 Hybrid 2: Use of an ANN for Peak Detection

With this approach, ANNs are used for ‘thresholding’ the accumulator array to determine peaks. The input to the network is the accumulator array after the Hough transform has been applied but no post processing of any form has been done. Number of input nodes is therefore 29760, the number of elements in the accumulator array. The output is a binary accumulator array indicating peaks. Number of output nodes is also 29760.

The programme developed to implement this approach crashed when setting up link 34,972,696 for a 1000-node-middle-layer system. It succeeded in setting up all the links with 100 nodes in the middle layer, but generates overflow errors when back-propagating errors after the first pass. It is likely that this behaviour is due to the neural network size being simply too large for the machine in use to handle.

The approach is aborted as infeasible with the combination of the size of accumulator array and the hardware of the machine in use for this work. In Chapter 8, some suggestions are made which might make this approach feasible.

7.1.3 Hybrid 3: Use of an ANN to Map from Peaks in Binary Accumulator Array to Vanishing Point

This approach aimed at taking a binary accumulator array as input, and giving out an indication of the position of the primary vanishing point. A binary accumulator array is the accumulator array with an entry of 1 for peaks and 0 for all other elements.

Training could not be achieved with a maximum of 200,000 attempts with various network topologies and values for learning rate and momentum term, and the approach was aborted as infeasible.

7.2 Hybrids with the Hough Transform in Artificial Neural Networks System

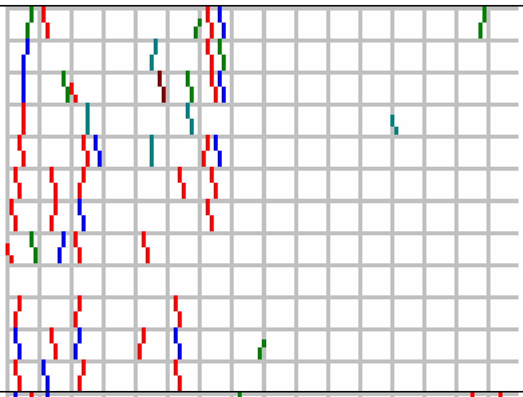
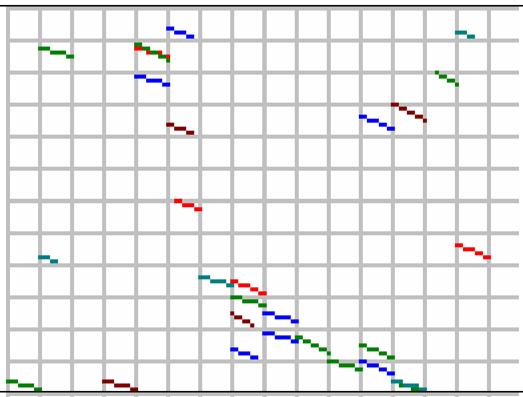
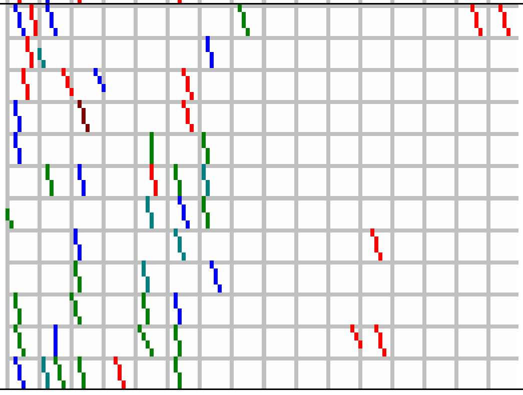
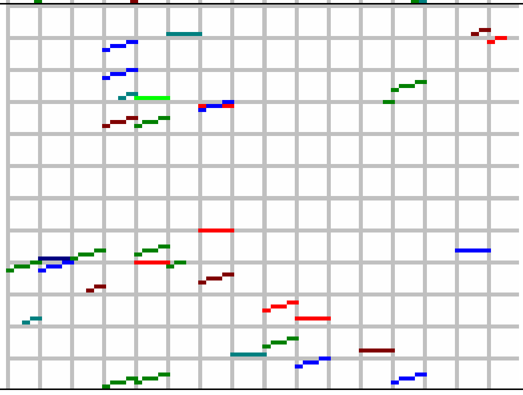
Hybrid possibilities identified in this category are discussed in *7.2.2 Hybrid 5: Use ANNs to find lines in Sub-Images and then use Hough transform to establish ‘full-*

picture' and *7.2.2 Hybrid 5: Use ANNs to find lines in Sub-Images and then use Hough transform to establish 'full-picture'*.

7.2.1 Hybrid 4: Use the Hough transform to find line segments in sub-images and then use ANN to establish full picture

This approach aimed to use the Hough transform to find line segments in sub-images (i.e. to replace the first stage of the artificial neural network approach of *6.3 Vision for Navigation based on Sub-Images Processing*), and then to use ANNs to establish results for the complete image.

Table 7.1 shows lines found by applying Hough transform to sub-images for different categories for the image in figure 6.1. They would constitute input for the second stage network of the neural network system of 6.3.

Category	Lines Found from HT in Sub-Images	Category	Lines Found from HT in Sub-Images
Vertical		Horizontal Backslash	
Vertical Backslash		Horizontal	

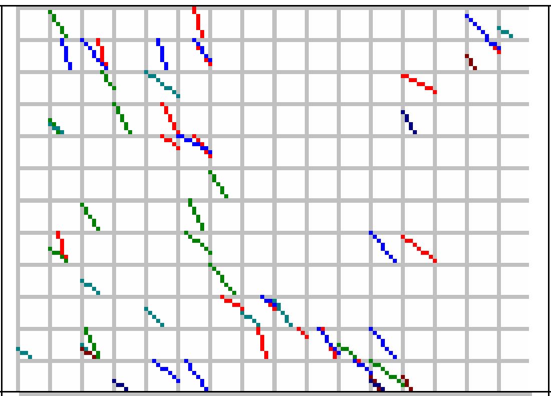
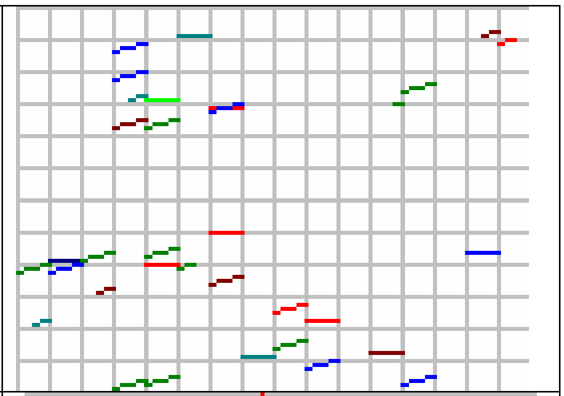
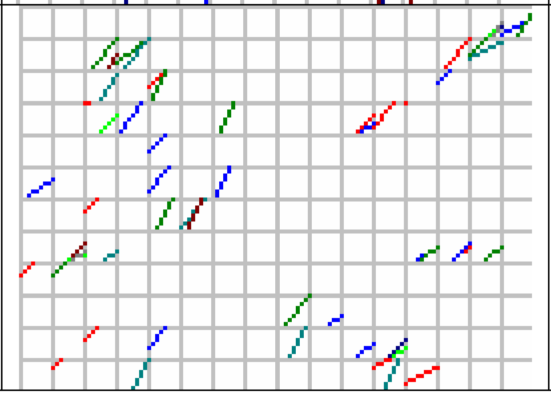
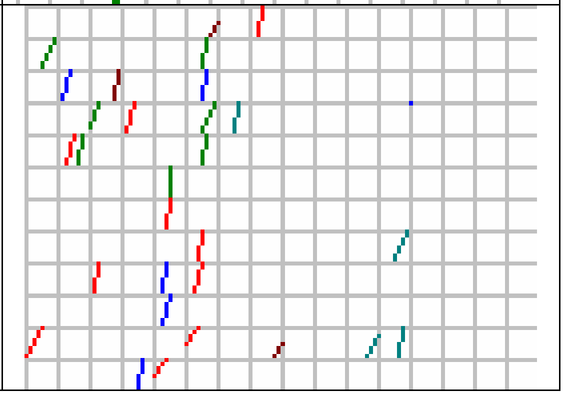
Backslash		Horizontal Slash	
Slash		Vertical Slash	

Table 7.1 Samples of lines found by applying Hough transform to sub-images for different categories

Although using the Hough transform to find line segments in sub-images is feasible, this hybrid method is not implemented because as discussed in *7.2.2 Processing 'Full Picture'*, using ANN to establish results for the complete image has not been successful.

7.2.2 Hybrid 5: Use ANNs to find lines in Sub-Images and then use Hough transform to establish 'full-picture'

In hybrid 5, the strategy is to use ANNs to find lines in sub-images 8 x 8 pixels in size, and then do a Hough transform on the results from sub images. Results from line recognition using ANN provide input for the Hough transform. Recall that the result from processing a sub-image using ANN as described in section *7.2 Vision based on Sub-Images Processing*, is stored as an 8 column binary vector. Each of the 8 elements of the vector says whether or not a line in each of the 8 categories of lines was found. There are 192 such vectors for every image corresponding to the 192 sub-images of the image. *7.2.1.9 Results* discusses these results in more detail.

The Hough transform in hybrid 5 is applied to consolidate results for each line category one category at a time. Each sub-image, represented by one of the 8 elements of its results vector, is taken as 1 pixel. As the values of θ are increased from 5° to 185°, for every sub-image, the input value considered for the transform is the element of the results vector corresponding to the category in whose range the current value of θ lies. The sub-sections which follow describe this process in more detail for each category. The discussions are based on processing for the image in figure 6.1 – the same one used in *7.2.1 Line Recognition and Categorisation from Sub-Images* and shown in figure 7.6.

7.2.2.1 Vertical Lines

The vertical lines found by passing sub-images of the image in figure 6.6 were shown in figure 6.7 superimposed on the input edge image in figure 6.6. They are shown by themselves in figure 7.1 below. Recall that the first element of the results vector indicates whether or not a vertical line was found in the sub-image.

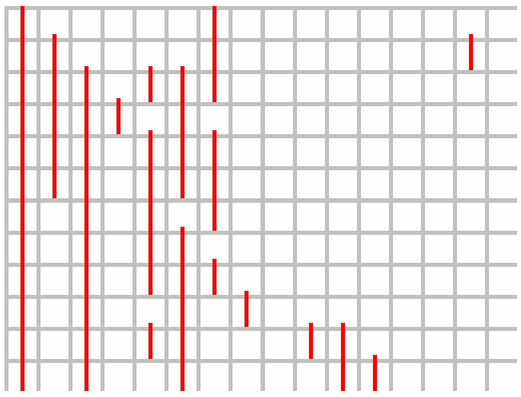


Figure 7.1 Vertical lines found in sub-images using ANNs

The Hough transform is applied for this category for θ between 175° and 184°. As discussed in Chapter 4, this range covers lines similar to lines in the range -5° to 4°. 175° to 184° is however, easier to deal with for setting up of the accumulator array (with all non-zero values) and processes such as application of the butterfly filter and selection of local maxima.

Significant lines are detected in a very similar way to detection of significant lines described in *5.1.6 Peak Detection*. First peaks are detected by application of a threshold automatically determined from a target number of peaks similar to the process described in *5.1.6.1 Threshold Application*. This target of 6 is selected by experimenting with images. The reduced butterfly filter is then applied. Recall that in *5.1.6.2 Application of the Butterfly Filter*, an idea of applying the filter only to peaks found from application of a threshold, to save processing time, was introduced. That idea is also used here. Local maxima are then selected from within 3 x 3 neighbourhoods in the butterfly filtered accumulator array. Note that this differs slightly from the case for processing Hough transform results from the complete image in *5.1.6.2 Application of the Butterfly Filter* where local maxima were selected from within 5 x 5 neighbourhoods. This is because the 'pixels' in the input image in this case are actually 8 x 8 pixel sub-images and for that reason, 'close' results are actually not that close (actual lines could be up to 15 pixels away).

Lines resulting from all these processes are illustrated in figure 7.2.

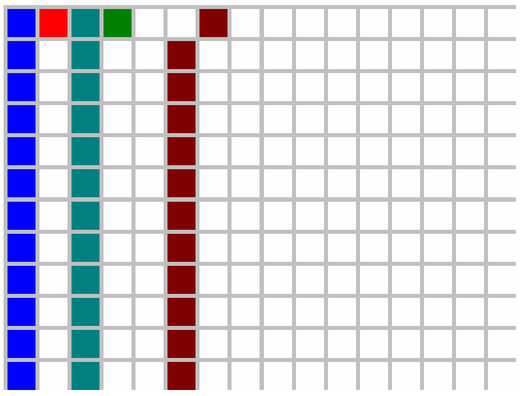


Figure 7.2 Vertical lines found in the 'full picture'

5 peaks were found in the example, differentiated in the figure by 5 different colours. Note that some of the lines overlap. Values of ρ and θ for the lines are shown in table 7.2.

Line ID	θ	ρ
0	175	8
1	180	8
2	175	6
3	180	6
4	175	3

Table 7.2 Parameter values for vertical lines found in the 'full picture'

The Hough transform finds lines but does not say where they begin and end. It also puts lines together which lie along the same infinite line. To determine endpoints of any possible actual lines on these infinite lines, further processing is done to determine which cells actually contributed to each line. This is done similar to what was described in *7.2.2 Determination of Actual Lines*. However, the details of the criteria for lines in this case are different. The criteria for a line with this hybrid approach are:

1. It must be at least 3 pixels long
2. It must be at least 1 pixel away from any other line on the same infinite line

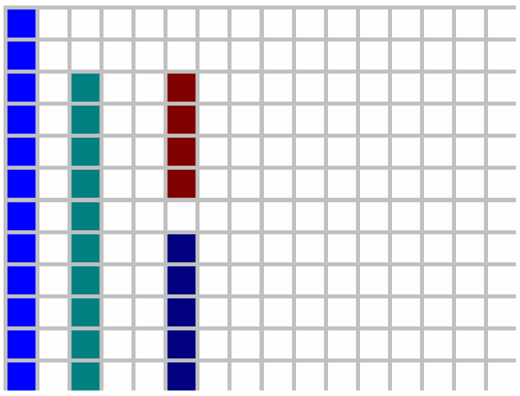


Figure 7.3 Vertical lines with valid sub-lines found in the 'full picture' with hybrid 5

Results are illustrated in figure 7.3. Some lines do not appear due to overlap. Sub-images making up each line are shown in table 7.3.

ID for Line with Valid Sub-Lines	Number of Sub-Lines	Number of Sub-Images in Sub-Line	IDs of Sub-Images in Sub-Line
0	1	12	0 16 32 48 64 80 96 112 128 144 160 176
1	1	12	0 16 32 48 64 80 96 112 128 144 160 176
2	1	10	34 50 66 82 98 114 130 146 162 178
3	1	10	34 50 66 82 98 114 130 146 162 178
4	2	4	37 53 69 85
		5	117 133 149 165 181

Table 7.3 Sub-Images which made up valid vertical sub-lines

7.2.2.2 Other Categories Lines

The results for the other categories were obtained in a similar manner and are summarised in tables 7.4, 7.5 and 7.6.

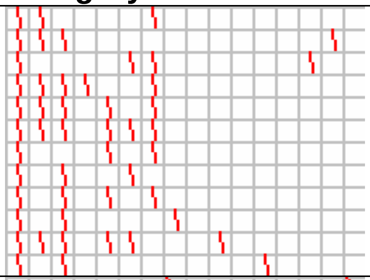
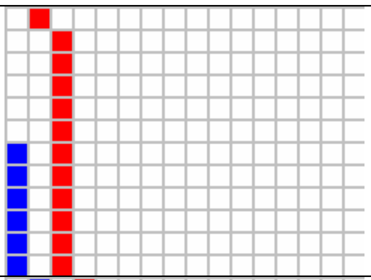
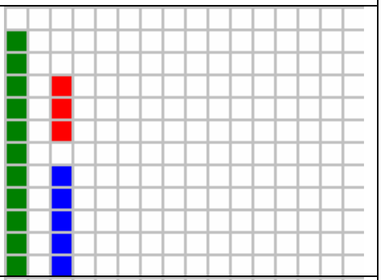
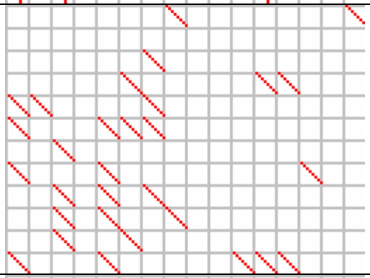
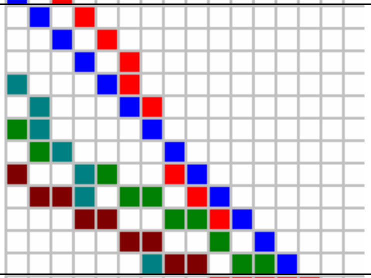
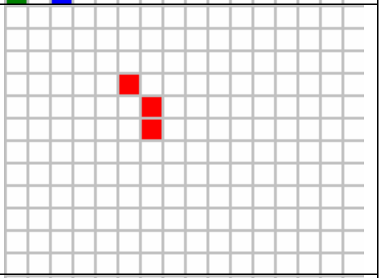
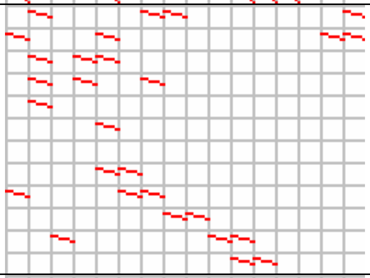
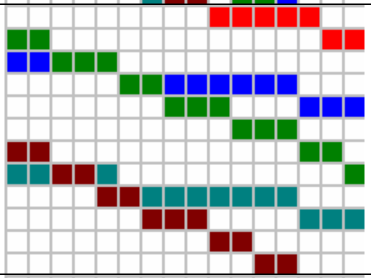
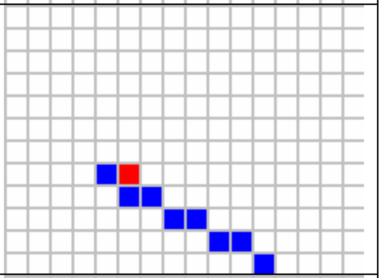
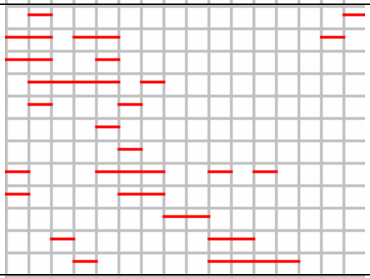
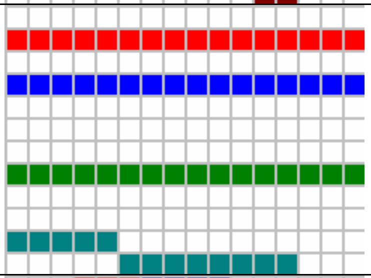
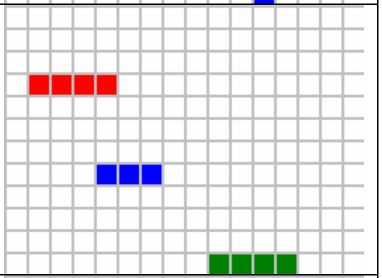
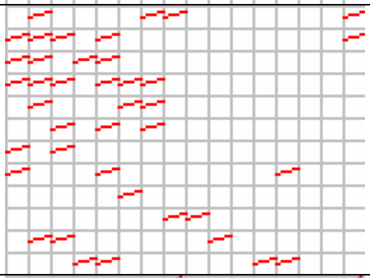
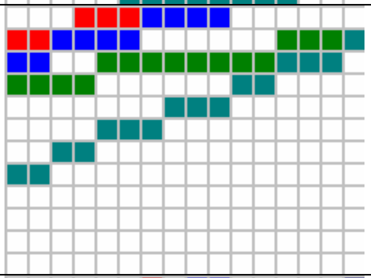
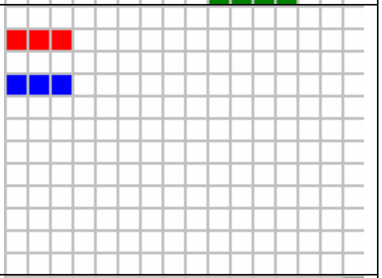
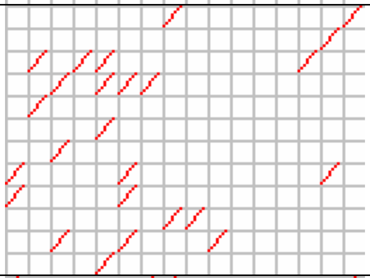
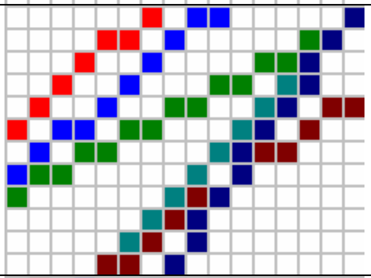
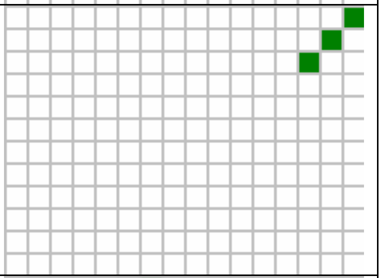
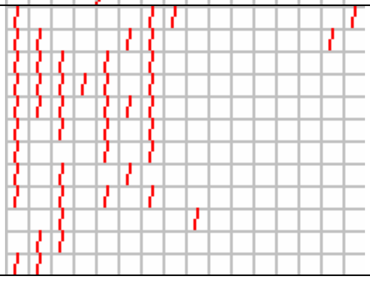
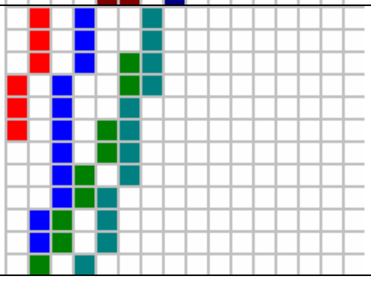
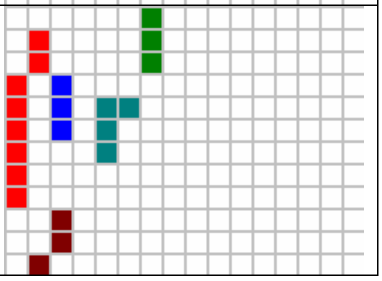
Category	Category Lines Found	Peaks Found	Valid Sub-Lines Found
Vertical Backslash			
Backslash			
Horizontal Backslash			
Horizontal			
Horizontal Slash			
Slash			
Vertical Slash			

Table 7.4 Examples of inputs to hybrid 5 Hough transform, peaks found and sub-lines found

Category	**θ Range for which Hough Transform was Performed**		**Number of Peaks Found**	**Parameters of Peaks Found**	
	Lower Bound	**Upper Bound**		θ	ρ
Vertical Backslash	5	24	2	5	-6
				5	-8
Backslash	25	64	5	30	-1
				45	-1
				60	-3
				35	-5
				60	-5
Horizontal Backslash	65	84	5	80	6
				80	3
				70	2
				80	-2
				65	-3
Horizontal	85	94	4	90	5
				90	3
				90	-1
				85	-5
Horizontal Slash	95	114	4	100	6
				105	6
				95	4
				110	2
Slash	115	154		130	6
				130	4
				115	2
				135	-1
				125	-2
				145	-2
Vertical Slash	155	174		170	8
				170	6
				155	4
				165	3

Table 7.5 Details of peaks found by hybrid 5 Hough transform

Category	Line ID with Valid Sub-Lines	Number of Sub-Lines	Number of Sub-Images in Sub-Line	IDs of Sub-Images in Sub-Line
Vertical Backslash	0	2	3	50 66 82
			5	114 130 146 162 178
	1	1	11	34 50 66 82 98 114 130 146 162 178
Backslash	0	1	3	53 70 86
Horizontal Backslash	0	1	4	116 117 133 134
	1	1	8	116 133 134 151 152 169 170 187
Horizontal	0	1	4	49 50 51 52
	1	1	3	116 117 118
	2	1	4	185 186 187 188
Horizontal Slash		1	3	16 17 18
		1	3	48 49 50
Slash	0	1	3	15 30 45
	1	1	3	15 30 45
	2	1	3	15 30 45
Vertical Slash	0	1	8	17 33 48 64 80 96 112 128
	1	1	3	50 66 82
	2	3	3	6 22 38
			4	68 69 84 100
			3	146 162 177

Table 7.6 Details of sub-lines found by hybrid 5 Hough transform

7.2.2.3 Combined Results

The full results of the Hough transform and subsequent post processing for all the line categories are presented together in figure 7.4.

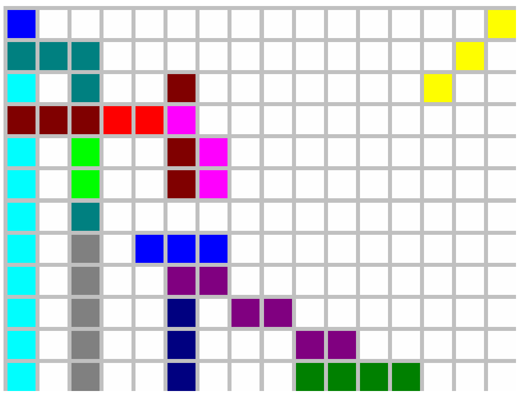


Figure 7.4 All valid sub-lines found in the 'full picture' with hybrid 5

7.2.2.4 Post Processing and Navigation

Post processing and navigation for this hybrid approach can be accomplished using adaptations of the approaches described for the Hough transform system discussed in *5.2 Post Processing and 5.4 Navigation* to the point of estimating the vanishing point and the corridor-door. The resolution with this approach is too poor for detection of wall-doors, so navigation into wall-doors is not feasible. Navigation along a corridor is.

7.3 Summary

A number of possible hybrid scenarios were studied, and the one found to be most feasible is the one which uses ANNs to find lines in sub-images and then consolidates the results from each of them together for the full image, by applying the Hough transform on the sub images as though they were pixels. This is discussed in detail in *7.3.2 Hybrid 5: Use ANNs to find lines in Sub-Images and then use Hough transform to establish 'full-picture'*.

Chapter 8 Summary, Conclusions and Suggestions for Future Work

8.1 Introduction to Chapter

The working objectives of the project by the time it was finished were:

1. Develop a mobile robot vision system based on line detection using the Hough transform
2. Develop a mobile robot vision system based on artificial neural networks which mimics some or all of the stages of the Hough transform based system
3. Look into the feasibility of a new system that is a hybrid of the two, which will draw from the strengths of both systems

The sub-sections which follow summarise what has been achieved. *8.2 Background and Preliminary Achievements* summarises preliminary and background work done and conclusions made while doing them. *8.3 Achievements towards a Mobile Robot Vision System based on Line Detection using the Hough transform*, *8.4 Achievements and Conclusions Related to a Mobile Robot Vision System based on Line Detection using Artificial Neural Networks* and *8.5 Achievements and Conclusions Related to a Hybrid Hough Transform/Artificial Neural Networks Mobile Robot Vision System* summarise what was achieved towards meeting objectives 1, 2 and 3 respectively, and conclusions made in the process.

8.2 Background and Preliminary Achievements

8.2.1 Environment

Prior to development of the systems, corridors within the school building were selected as constrained but realistic environments in which to test the systems to be developed. Issues which needed to be taken into account within corridor environments included the contrast between the floors and walls; varying lighting levels and their effects; the additional features which artificial light sources within the corridor introduce when they are switched on; reflections on the floor which constitute ‘additional features’; and the presence of doors and other objects within the corridor such as radiators.

8.2.2 Background

A background study of the key concepts of the project was undertaken. Major issues related to them are summarised as part of these thesis.

8.2.3 Related Work

Previous works related to the key concepts in these work were reviewed and summarised under certain headings including *Works Related to the Performance of the Hough Transform Algorithm*, *The Hough Transform in Vision Systems for Mobile Robot Navigation*, *Artificial Neural Networks in Vision Systems for Mobile Robot Navigation*, *Works which Employ a Combination of the Hough Transform*, and *Other Related Works*.

8.2.4 Pre-Processing

When an image has been captured by the robot’s camera, before any of the core algorithms are applied, it is necessary to pre-process it. In this work, pre-processing tasks used include resizing of the captured image to an optimal size which maintains necessary information without consuming more processing resources than necessary. The optimal size was selected by trying various image sizes. Other pre-processing tasks include edge-detection using the popular Sobel filters, and edge-thinning using a modified version of the thinning method of (Park, 2000) proposed by this work, which has been found to be more suitable.

In edge-detection, this work has proposed a method of automatic detection of the threshold for selection of significant edge-points, which is a function of the particular image under consideration rather than using a fixed threshold for all images.

This work has also analysed the costs and benefits of thinning to the Hough transform vision system developed in this work taking into account issues such as time taken for various subsequent processes and quality of results. The analysis done includes determination of what would amount to significant difference in processing time given the particular hardware used for the study. It concludes that a significant amount of time is saved when thinning is employed as part of pre-processing despite the time that the thinning process itself takes. It also concludes that the improvement in quality due to thinning is so significant it can actually affect the recognition rate of high level features such as doors and corridor edges, and consequently, the accuracy of navigation of the robot.

8.3 Achievements and Conclusions Related to a Mobile Robot Vision System based on Line Detection using the Hough transform

A vision system has been developed for a mobile robot based on line detection with the Hough transform. Several processes were carried out towards development of the system. They are summarised in the sub-sections which follow.

8.3.1 Customisation and Implementation of the Hough Transform

Towards customisation and implementation of the Hough transform, tasks carried out include choice of origin for image space, choice of resolution for the parameters of the transform based on analysis of the input images and the nature of the output required, and determination of the range of the parameters to be used for the transform. The Hough transform has been applied using the polar form of the equation of a straight line.

This work has developed and implemented a scheme for peak detection which includes automatic determination of the most appropriate threshold for the image under consideration, application of the reduced butterfly filter of (Boyce et al 1987) to entries above the threshold found only, and selection of elements of the accumulator array that are local maxima within a 5 x 5 neighbourhood. This scheme has been found to be very efficient and accurate.

By way of post-processing, a scheme has been developed to determine endpoints for genuine sub-lines from the original image while eliminating ‘false’ lines which emerged by accumulating votes from random points by ‘coincidence’. A group of contributing points constitutes a valid sub-line if it meets two criteria – there are at least a certain number, $L_{\min}$ of points in the group, and the separation between the group and any other points is at least $S_{\min}$ pixels. Values of $L_{\min}$ and $S_{\min}$ were determined by analysis of the nature of the images in this work. Results from this scheme have been very good.

8.3.2 Vanishing Point Estimation

The determination of high-level features in this work relies heavily on knowledge of the position of the primary vanishing point on the image. A scheme has been developed for determination of the primary vanishing point which involves finding intersection points for all pairs of actual lines found and awarding them points based on a scheme developed by this work. The intersection point that scores the highest using this scheme is returned as the vanishing point, provided the total of points it has scored has reached a certain threshold which was determined empirically a priori. If the threshold is not reached by any of the intersection points, there is not enough evidence to confidently point out the vanishing point.

This scheme has performed very well.

8.3.3 Line Categorisation

In order to facilitate determination of high-level features, this work proposed a categorisation scheme which categorises lines into vertical, vertical-backslash, backslash, horizontal-backslash, horizontal, horizontal-slash, slash, and vertical-slash categories depending on the angle of the line. It also assigns a magnitude and a sign to the line depending on its distance from the vanishing point.

This categorisation has proved to be very helpful in post-processing.

8.3.4 High-Level Features Determination

Corridor recognition is critical to successful navigation in a corridor. In this work, this has been achieved by searching for a left-corridor-edge and a right-corridor-edge using a points-based scheme developed for the purpose. To select a left-corridor-edge for example, points are assigned to lines according to whether or not they fit a certain criterion - whether they are in the range 90° to 180° - and if so, how they score against three further criteria. The three criteria relate to the distance of the most south-westerly point on the lines to the bottom-left corner of the image, the shortest distance of the lines to the primary vanishing point on the image if it has been found, and which categories the lines belong to (using the categorisation scheme developed by this work described earlier). The line which scores the highest is the left-corridor-edge provided its score is equal to or higher than a certain cut off score determined empirically.

Similar schemes have been developed to detect doors, whether they are at the end of a corridor, or in a wall.

These schemes have worked well.

8.3.5 Navigation

A navigation scheme has been designed to work with the outcome of post-processing and high-level features detection, and the navigation objective of the robot. Three simple navigation objectives were developed – move-along-corridor, navigate-into-door-on-the-left, and navigate-into-door-on-the-right. These simple objectives can be combined in any way to derive more sophisticated and more useful objectives.

This scheme has not been tested exhausted due to time constraint.

8.4 Achievements and Conclusions Related to a Mobile Robot Vision System based on Line Detection using Artificial Neural Networks

Attempts were made to develop another vision system based on line detection using artificial neural networks. Two approaches were investigated.

One of them attempts to find lines belonging to the categories defined in this work. It looks for lines in each category with a separate neural network trained to recognise lines in that category in parallelograms extracted from the image of with shapes with dimensions dictated by the range of the angle lines in the category take. Lines found would then be analysed for existence of high level features. This approach worked quite well for the vertical category in which lines tend to be very long. It failed woefully for a slanted category for which lines tend to be very short relative to the stripes that contain them. The approach was aborted.

The second approach involved breaking an image down to sub-images, and then determining whether the sub-image contains a line in any of the categories set up in this work using a group of 8 neural networks called the stage 1 networks. Results from these stage 1 networks are then passed to a stage 2 network which is set up to work out the direction the robot should move in. The stage 1 networks did reasonable well in recognising the target lines, but the stage 2 network failed to properly recognise the correct direction the robot should take for most test images. This means the system as a whole did not do well. However, the stage 1 networks which did do well were helpful in the development of a hybrid Hough transform/neural network vision system.

8.5 Achievements and Conclusions Related to a Hybrid Hough Transform/Artificial Neural Networks Mobile Robot Vision System

New approaches were considered which attempt to either bring ANNs into a system originally designed to use the Hough transform or bring the Hough transform into a system originally designed to use ANNs. The system that was most successful used artificial neural networks to detect valid sub-lines albeit with reduced resolution, and then used the Hough transform to estimate the actual lines the sub-lines came from. The system is capable of supporting navigation along a corridor but not into doors because the resolution involved is too low to distinguish the edges of such doors.

8.6 Summary of Suggestions for Future Work

Although a navigation scheme was proposed for the Hough transform vision system developed in this work, it was not adequately tested. It would be beneficial if it were properly tested, and if necessary, adjusted so that the navigational accuracy of the system could be evaluated. Navigation schemes could also be developed for other vision systems discussed in this work to facilitate evaluation and comparisons of the various systems.

It was initially hoped that this work would include a detailed comparison of the various vision systems discussed in this work. However, this was not accomplished as other areas of the work grew beyond what was originally anticipated, and there was not enough time to do such comparison. It would be interesting if the systems discussed in this work, and possibly others, were compared. This could cover issues such as time taken to run, accuracy in feature detection, resulting navigational accuracy, processor and memory usage, etc.

Attempts to imitate some processes in the Hough transform with artificial neural networks failed because of memory requirements. It might be worth attempting them with smaller versions of the accumulator array, or machines with more memory.

Neural networks for the purpose of line detection in this work were applied to cells which were either square cells with optimal sizes for detecting valid sub-lines, or parallelograms which contain full line segments. It could be more accurate, although more time consuming if the search parallelograms and cells were to glide with reduced intervals of the order of about a single pixel. It would be interesting to document and study the changes in accuracy and resource consumption that this would bring about.

References

1. Anonymous. Neural Networks [Web Page]. Accessed 2007 Jun 7. Available at: http://en.wikipedia.org/wiki/Neural_network.

2. Atiquzzaman M . Multiresolution Hough Transform - An Efficient Method of Detecting Patterns in Images. IEEE Transactions on Pattern Analysis and Machine Intelligence. 1992; 14(11):1090-1095.

3. Banjanovic-Mehmedovic, Lejla; Petrovic, Ivan and Ivanjko, Edouard.2004. Hough transform based Correction of Mobile Robot Orientation. IEEE Conference on Industrial Technology. p1573-1578.

4. Boyce J. F., Jones G. A. and Leavers V. F. (Wheatstone Laboratory, Kings College, Strand, London, WC2R 2LS, U. K.). An Implementation of the Hough transform for line and circle location. SPIE Inverse Problems in Optics; The Hague, Netherlands. Bellingham, Washington USA: SPIE-The International Society of Optical Engineering; 1987; c1987.

5. Crochat, Philippe and Franklin, Daniel. Back-propagation Neural Network Tutorial [Web Page]. Accessed 2007 Jun 25. Available at: http://pcrochat.online.fr/webus/tutorial/BPN_tutorial4.html.

6. Chang, Tsai-Yu and Hsu Jane Yung-jen (Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan). Proceedings of the 1994 International Symposium on Artificial Neural Networks. International Symposium on Artificial Neural Networks: 701-707.

7. Davies, E. R. 1997. Machine Vision: Theory, Algorithms and Practicalities. 2nd Ed. Academic Press. San Diego.

8. Dempsey, G. L. and McVey E. S. A Hough Transform System based on Neural Networks. IEEE Transaction on Industrial Electronics. 1992; 39(6):522-528.

9. Dempsey, G. L. and McVey E. S. Hough Transform System Enhancement Resulting from Neural Network Implementation. Proceedings of the IEEE Southeastcon Conference. Williamsburgh, VA. Vol. 1, April 1991. 94-98.

`10. Djekoune, A. O. and Achour, K. 2000. Visual Guidance Control based on the Hough Transform. IEEE Intelligent Vehicles Symposium.

11. Duda and Hart. 1973. Pattern Classification and Scene Analysis. Joh Wiley and Sons. New York.

12. Espinos, Cecilia and Perkowski, Marek A. 1991. Hierarchical Hough Transform for Vision of the Psubot.

13. Forsberg, Johan and Ulf Larson and Ake Wernerson. Mobile robot navigation using the range-weighted Hough transform. IEEE Robotics and Automation Magazine. 1995; 2(1):18-26.

14. Grimson, W. Eric L. and Huttenlocher Daniel P. On the Sensitivity of the Hough Transform for Object Recognition. IEEE Transactions on Pattern Analysis and Machine Vision. 1990; 12(3):255-274.

15. Hough, Paul, inventor. Method and Means For Recognising Complex Patterns. United State of America 3069654. 1962.

16. Inigo, Rafael M and Torres Raul E. Mobile robot navigation with vision based neural networks. Mobile Robots IX. 1995; 2353:68-79.

17. Jagadeesan, Ananda. PhD Thesis. Aberdeen: Robert Gordon University; 2006.

18. Leavers, V. F. Shape Detection in Computer Vision Using the Hough Transform. London: Springer-Verlag; 1992.

19. Li, Wei; Jiang, Xiaojia and Wang, Yangxing. Recognition for Vision Navigation of an Autonomous Vehicle by Fuzzy Reasoning. Science Direct Fuzzy Sets and Systems. 1998. vol 93, issue 3. p275-280.

20. Low, Adrian. Introductory Computer Vision and Image Processing. London: McGraw-Hill Book Company; 1991.

21. Meng, Min and Kak A. C. (Robot Vision Laboratory, Purdue University, W. Lafayette). IEEE Conference on Robotics and Automation; USA. 1993: 750-757.

22. Orr, Genevieve and Schraudolph, Nici and Cummins Fred. Neural Networks Lecture Notes [Web Page]. 1999; Accessed 1925 Jun 7. Available at: http://www.willamette.edu/~gorr/classes/cs449/brain.html.

23. Park, Jung-Me and Chen, Hui-Chuan and Huang Shu T. (Dept of Computer Science, Alabama Univ, Tuscaloosa, AL, USA.). A new gray level edge thinning method. Proceedings of the ISCA 13th International Conference. Computer Applications in Industry and Engineering; Honolulu, HI, USA. Cary, NC, USA: International Society for Computers & Their Application. ISCA; 2000.

24. Princen J, Illingworth J and Kittler J. Hypothesis Testing: A Framework for Analyzing and Optimizing Hough Transform Performance. IEEE Transaction on Pattern Analysis and Machine Intelligence. 1994. Vol 16 Issue 4. p329-341.

25. Rogers J. Object-Oriented Neural Networks in C++. Morgan Kaufmann; Book & Disk 1st edition. 1996.

26. Smith S M and Brady J M (Defence Research Agency, Chobham Lane, Surrey, UK). SUSAN - a new approach to low level image processing TR95SMS1. 1995.
Notes: Technical papar by same author: Int. Journal of Computer Science, 1997. Same title as report

27. Sonka, Milan and HLavac, Vaclac and Boyle Roger. Image Processing, Analysis and Machine Vision . London: PWS Publishing; 1999.

28. Tyson, Jonathan. Robot Control from Video Images. Honours Project, Robert Gordon University. 1995.

29. Vaughn, David L and Arkin Ronald C. Workstation Recognition using a Constrained Edge-based Hough Transform for Mobile Robot Navigation. 1990.

30. Yun, Xiaoping; Latt; Khine and Glennon J Scott. Mobile Robot Localization using the Hough Transform and Neural Networks. IEEE International Symposium on Intelligent Control. 1998.

Appendix A Publication Produced During Research

Damaryam, G., & Dunbar, G. (2005). A Mobile Robot Vision System for Self Navigation using the Hough Transform and Neural Networks. In the Proceedings of EOS Conference on Industrial Imaging and Machine Vision, Munich, 13 - 15 June, pp. 72.

Appendix B Details of Results of Sub-Lines Determination for a Sample Image

Line ID	Number of Sub-Lines	Number of Points in Sub-Line	IDs of Contibuting Points for Sub-Line
0	1	17	1781 1909 2037 2165 2293 2421 2549 2677 2806 2934 3062 3190 3318 3446 3574 3702 3830
1	1	9	9585 9842 9970 10098 10226 10354 10482 10610 10738
2	2	11	202 330 459 716 845 973 1102 1230 1359 1487 1616
		14	2773 2902 3030 3159 3416 3544 3673 3801 3802 4059 4187 4316 4573 4701
3	1	10	5078 5206 5334 5462 5591 5719 5847 5975 6103 6231
4	1	8	5078 5206 5334 5462 5591 5719 5847 5975
5	1	10	3019 3147 3275 3403 3532 3660 3916 4044 4173 4301
6	2	12	2506 2634 2891 3019 3147 3275 3532 3660 3916 4044 4173 4301
		9	8531 8787 8915 9043 9171 9428 9556 9684 9812
7	1	21	4571 4572 4573 4575 4576 4578 4580 4581 4582 4583 4585 4586 4587 4589 4590 4592 4594 4595 4597 4598 4599
8	2	8	1867 2123 2251 2380 2508 2636 2764 2892
		10	9552 9681 9809 9937 10193 10321 10449 10577 10833 11089
9	3	12	2001 2129 2257 2385 2513 2641 2769 2897 3025 3153 3281 3409
		34	3793 3921 4049 4177 4305 4433 4561 4689 4817 4945 5073 5201 5329 5457 5585 5713 5841 5969 6097 6225 6353 6481 6609 6737 6865 6993 7249 7377 7505 7633 7761 7889 8017 8145
		20	9169 9297 9425 9553 9681 9809 9937 10193 10321 10449 10577 10833 11089 11217 11345 11473 11601 11729 11857 11985
10	2	36	2772 3028 3156 3284 3412 3540 3668 3796 3924 4052 4180 4308 4436 4564 4692 4820 4948 5076 5204 5332 5460 5588 5716 5844 5972 6100 6228 6356 6484 6612 6740 6868 6996 7124 7252 7380
		12	9428 9556 9684 9812 9940 10068 10196 10324 10452 10580 10836 10964
11	2	30	8652 8654 8773 8774 8775 8777 8779 8896 8897 8898 8899 9017 9018 9019 9020 9021 9022 9140 9141 9142 9143 9144 9261 9263 9265 9266 9384 9385 9386 9387

		23	9505 9506 9507 9508 9509 9625 9626 9627 9628 9629 9630 9631 9747 9748 9749 9750 9751 9752 9869 9870 9871 9872 9873
12	2	10	5719 5847 5975 6103 6231 6359 6487 6615 6743 6871
		9	9428 9556 9684 9812 9940 10068 10196 10324 10452
13	1	11	5591 5719 5847 5975 6103 6231 6359 6487 6615 6743 6871
14	1	14	9626 9627 9628 9629 9630 9631 9632 9746 9747 9748 9749 9750 9751 9752
15	1	29	9280 9281 9282 9283 9401 9403 9404 9405 9406 9523 9525 9526 9527 9645 9646 9647 9649 9766 9767 9768 9769 9770 9771 9889 9890 9891 9892 10012 10015
16	1	17	7260 7388 7516 7644 7772 7900 8028 8156 8284 8412 8668 8796 8924 9052 9180 9308 9564
17	1	12	9765 9766 9767 9768 9769 9770 9771 9772 9889 9890 9891 9892
18	1	10	10071 10327 10455 10583 10711 10839 10966 11222 11350 11478
19	2	12	4573 4701 4957 5085 5213 5341 5469 5597 5725 5853 5981 6109
		19	8028 8156 8284 8412 8668 8796 8924 9052 9180 9308 9564 9692 9820 9948 10076 10204 10332 10460 10588
20	1	13	5994 6122 6250 6378 6506 6635 6763 6891 7019 7147 7275 7403 7531
21	1	8	6635 6763 6891 7019 7147 7275 7403 7531
22	1	8	246 501 629 757 885 1140 1268 1396
23	1	8	759 887 1015 1143 1270 1398 1526 1654
24	1	8	7282 7410 7538 7666 7794 7922 8050 8178
25	2	15	6386 6514 6642 6770 6898 7026 7154 7282 7410 7538 7666 7794 7922 8050 8178
		8	9842 9970 10098 10226 10354 10482 10610 10738
26	1	11	2806 2934 3062 3190 3318 3446 3574 3702 3830 4086 4214
27	1	10	3832 4088 4216 4344 4472 4727 4855 4983 5111 5367
28	1	19	2806 2934 3062 3190 3318 3446 3574 3702 3830 4086 4214 4342 4470 4598 4855 4983 5111 5367 5495
29	1	19	2810 2938 3066 3194 3322 3450 3578 3834 3962 4090 4218 4474 4602 4730 4986 5114 5242 5498 5626

Appendix C Artificial Neural Network Training Set and Sample Test Results for Vertical Category

Training Set for Vertical Line Recognition

```
0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0

1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1
0 0 0 0 1 1 0 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0

2
0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 0 1 0 0 0 0 0
1 0 1 0 0 0 0 0
1 0 0 1 0 0 0 0
1 0 0 0 0 0 0 0
0 1 0 0 1 0 0 0
0

3
0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 1 1 0 0
0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 0
0

4
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 1 1 0 0 1
1 1 1 1 0 0 1 0
0 0 0 0 0 0 0 0
0

5
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 1 0
1 1 1 0 0 1 0 0
0 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0
0

6
0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0

7
0 0 0 0 0 0 0 0
0 0 1 0 1 1 0 0
0 0 0 0 1 0 0 0
0 1 0 1 0 0 0 0
0 1 0 1 0 1 0 0
0 0 0 1 0 1 0 0
1 0 1 0 0 1 0 1
1 0 1 0 0 1 0 1
1

8
0 0 0 0 0 0 0 0
1 1 0 0 0 0 0 0
1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1

9
0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0
0 0 1 0 1 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 1 0 0
0 0 0 1 0 1 0 0
1

10
0 1 0 0 1 0 0 0
0 1 0 0 1 0 0 0
0 0 1 0 1 0 1 0
0 0 0 0 1 0 1 0
0 0 1 0 1 0 1 0
0 0 1 0 1 0 1 0
0 0 1 0 1 0 1 0
0 0 1 0 1 0 1 0
1

11
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 1
0

12
0 0 1 0 0 1 0 1
0 1 0 0 0 1 0 1
0 1 0 0 0 1 0 1
0 1 0 0 0 1 0 1
1 1 0 0 0 1 0 1
0 1 0 0 0 1 0 1
0 0 1 0 0 1 0 1
0 0 1 0 0 1 0 1
1

13
0 0 0 1 0 1 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 1 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0

14
0 0 1 0 0 0 0 0
0 0 1 0 1 0 0 0
0 0 0 0 1 0 0 0
0 0 1 0 1 0 1 0
0 0 1 0 1 0 1 0
0 0 1 0 1 0 1 0
0 0 1 0 1 0 0 1
0 0 1 0 1 0 0 1
1

15
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
0 1 0 0 0 0 1 0
0 1 0 0 0 0 1 1
0 0 0 0 0 0 0 1
1

16
0 0 0 1 1 0 0 0
1 1 1 1 0 1 0 0

17
0 0 0 0 0 0 0 0
1 1 1 1 1 0 0 0

18
0 0 0 0 0 1 0 1
0 0 0 0 0 0 0 1

19
0 0 1 0 0 1 0 1
0 0 1 0 0 1 0 1
```

```
0 0 0 0 0 0 0 0      0 0 0 0 1 0 0 0      0 0 0 0 1 0 1 0      0 0 1 0 0 1 0 1
0 0 0 0 0 0 0 0      0 0 0 0 0 1 0 0      0 0 0 0 0 0 1 0      0 1 0 0 0 1 0 1
0 0 0 0 0 0 0 0      0 0 0 0 0 1 0 0      0 0 0 1 0 0 1 0      0 0 1 0 0 1 0 1
0 0 0 0 0 0 0 0      0 0 0 0 0 1 0 0      0 0 0 1 0 1 0 0      0 0 0 0 0 1 0 1
0 0 0 0 0 0 0 0      0 0 0 0 1 0 0 0      0 0 1 0 0 1 0 0      0 0 1 0 0 1 0 1
1 1 1 0 0 0 1 1      0 0 0 0 1 0 0 0      0 0 1 0 1 0 0 1      0 0 1 0 0 1 0 1
0                    0                    0                    1

20                   21                   22                   23
0 0 1 0 1 0 0 1      0 0 1 0 0 0 0 0      0 0 0 1 1 1 1 1      1 1 1 1 0 0 0 0
0 1 0 0 1 0 0 1      1 0 1 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 1 0 0 1 0 1 0      1 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 1 0 0 1 0 0 0      0 1 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 1 0 0 1 0 1 0      0 0 0 0 0 0 0 0      0 1 1 1 0 0 1 0      1 1 1 1 0 0 0 0
0 1 0 0 1 0 1 0      0 1 0 0 0 0 0 0      1 0 0 0 1 1 0 0      0 0 0 0 0 0 0 0
0 1 0 1 0 0 1 0      0 0 1 0 1 0 0 1      1 0 0 0 0 0 0 0      0 0 0 0 1 0 0 0
0 1 0 1 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 1 0 0 1
1                    0                    0                    0

24                   25                   26                   27
0 0 0 0 1 0 0 0      0 0 1 0 0 1 0 0      0 0 0 0 0 0 0 0      0 1 0 1 0 0 1 0
0 0 0 0 1 0 0 1      0 0 1 0 0 1 0 1      1 0 0 0 0 0 0 0      0 1 0 1 0 0 0 1
0 0 0 0 1 0 0 1      0 0 1 0 0 1 0 0      0 0 0 0 0 0 0 0      0 1 0 1 0 0 1 0
0 0 0 0 1 0 0 1      0 0 1 0 0 1 0 0      0 0 0 0 0 0 0 0      0 1 0 1 0 0 0 0
0 0 0 0 1 0 0 0      0 0 1 0 0 0 0 1      0 0 0 0 0 0 0 0      0 1 0 1 0 1 0 0
0 0 0 0 1 0 0 0      1 1 0 0 0 1 0 1      0 0 0 0 0 0 0 0      0 1 0 1 0 1 0 1
1 0 1 0 1 0 0 0      0 0 0 0 0 1 0 1      0 0 0 0 0 0 0 0      0 1 0 1 0 1 0 0
0 0 1 0 0 0 0 0      0 0 0 0 0 1 0 1      0 0 0 0 0 0 0 0      1 0 0 1 0 1 0 0
1                    1                    0                    1

28                   29                   30                   31
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 1 0 0 0
0 0 1 1 0 1 1 1      1 1 0 0 0 0 0 0      0 0 1 1 0 1 1 1      0 1 0 0 1 0 0 0
0 0 0 1 0 0 0 0      0 1 1 1 1 1 1 1      1 1 0 0 1 0 0 0      1 1 0 0 0 0 0 0
0 0 0 0 0 0 1 0      0 0 0 0 0 0 0 0      0 1 0 0 0 0 0 0      0 1 0 0 1 1 0 1
0 0 0 1 0 0 0 0      0 0 0 1 0 1 0 0      0 0 0 0 1 1 1 0      0 1 0 0 0 0 0 1
0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0      0 1 1 0 1 0 0 0      0 1 0 0 1 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 1 0 0 0 0 0 0      0 1 0 0 0 0 0 0
0 1 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 1 0 0 0 0 0 0      0 1 0 0 0 1 0 1
0                    0                    1                    1

32                   33                   34                   35
0 0 0 0 0 1 0 1      0 0 0 0 0 0 0 0      0 1 0 1 0 1 0 0      0 1 0 1 0 0 0 0
0 0 0 0 0 1 0 1      1 0 0 0 0 0 0 0      0 0 0 1 0 1 0 0      0 1 0 0 0 1 0 0
1 1 1 0 1 0 0 0      0 0 0 0 0 0 0 0      0 1 0 1 0 0 1 0      0 1 0 1 0 0 0 0
0 0 1 0 1 0 0 0      1 0 0 0 0 0 0 0      0 1 0 1 0 0 1 0      0 1 0 1 0 0 0 0
0 0 1 0 0 1 0 0      1 0 0 0 0 0 0 0      0 0 0 1 0 0 1 0      0 1 0 1 0 0 0 0
0 0 1 1 0 1 0 0      1 0 0 0 0 0 0 0      0 0 0 1 0 0 1 0      0 0 0 1 0 0 0 0
1 0 1 0 0 1 0 0      1 0 0 0 0 0 0 0      0 1 0 1 0 0 0 0      0 0 0 0 0 0 0 0
0 0 1 0 0 1 0 0      1 0 0 0 0 0 0 0      0 1 0 1 0 1 0 0      0 0 0 0 0 0 0 0
1                    1                    1                    1

36                   37                   38                   39
```

```
0 1 0 0 1 0 0 0     0 1 0 1 0 1 0 1     0 0 1 0 0 0 0 0     0 0 0 0 0 0 0 0
0 1 0 0 1 0 0 0     0 1 0 1 0 1 0 1     0 0 1 0 0 1 0 0     0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0     0 1 0 0 0 1 0 1     0 1 0 0 0 1 0 0     1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 1 0 1 0 1 0 1     0 1 0 0 0 1 0 0     1 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0     0 1 0 1 0 0 0 1     0 1 0 0 0 1 0 0     0 0 0 0 0 0 0 0
0 1 1 0 0 0 0 0     0 1 0 0 0 1 0 0     0 0 0 0 0 1 0 1     0 0 0 0 0 0 0 0
0 1 0 0 1 0 0 0     0 1 0 0 0 0 0 0     0 0 0 0 0 1 0 0     1 0 0 0 0 0 0 0
0 1 0 0 1 0 0 0     0 1 0 0 1 1 0 0     0 0 0 0 0 1 0 0     0 0 0 0 0 0 0 0
1                   1                   1                   0

40                  41                  42                  43
0 0 0 0 0 0 0 0     1 0 0 1 0 1 0 0     1 0 1 0 1 0 0 0     0 1 0 0 1 0 0 0
0 0 0 0 0 0 0 0     0 1 0 1 0 1 0 1     1 0 1 0 0 0 0 0     0 1 0 0 1 0 0 0
0 0 0 0 0 0 0 0     0 1 0 1 0 1 0 1     0 0 1 0 0 0 0 0     0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 1 0 1     0 0 0 0 0 0 0 0     0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0     1 0 0 1 0 1 0 1     0 0 1 0 0 0 0 0     0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 1     0 0 1 0 1 0 0 0     1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1     0 1 0 1 0 0 0 0     0 0 1 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1     0 1 0 1 0 0 0 0     0 0 1 0 0 0 0 0     0 1 0 0 0 0 0 0
0                   1                   1                   1

44                  45                  46                  47
0 1 0 0 0 0 0 0     0 0 0 0 0 1 0 0     0 0 0 0 0 0 0 0     0 0 0 1 0 0 0 0
0 1 0 0 1 0 0 0     0 0 0 0 0 1 0 0     0 0 0 0 0 0 0 0     0 1 0 0 0 0 0 0
0 1 0 0 0 0 0 0     0 0 0 0 0 1 0 0     1 0 0 0 0 0 0 0     1 0 0 1 0 0 0 0
0 1 0 0 0 0 0 0     0 0 0 0 0 1 0 0     1 0 0 0 0 0 0 0     1 0 0 1 0 0 0 0
0 1 0 0 0 0 0 0     0 0 0 0 0 1 0 0     0 0 0 0 0 0 0 0     0 1 0 1 0 0 0 0
0 1 0 0 0 0 0 0     0 0 0 0 0 1 0 0     1 0 0 0 0 0 0 0     0 1 0 1 0 0 0 0
0 1 0 0 0 0 0 0     0 0 0 0 0 0 0 0     1 0 0 0 0 0 0 0     0 0 0 1 0 0 0 0
0 1 0 0 0 0 0 0     0 0 0 0 0 1 0 0     1 0 0 0 0 0 0 0     1 0 1 0 0 0 0 0
1                   1                   1                   1

48                  49                  50                  51
0 0 1 0 0 0 0 0     0 1 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 1 0 0 0 0 0     0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0     0 1 0 0 0 0 0 0     0 0 1 0 0 0 0 0     0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0     0 1 0 0 0 0 0 0     0 1 0 0 0 0 0 0     0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0     0 0 0 0 0 0 0 0     0 1 0 0 0 0 0 0     0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 1     0 1 0 0 0 1 1 0     0 1 0 0 0 0 0 0     0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 1     0 1 1 1 1 1 1 1     0 1 0 0 0 0 0 0     0 0 0 0 0 1 0 1
0 0 1 0 0 0 0 0     0 1 0 0 0 0 0 1     0 1 0 0 0 0 0 0     0 0 0 0 1 0 0 0
1                   1                   1                   1

52                  53                  54                  55
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 1 0
1 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 1 0 1 1 0 1
1 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     1 1 1 0 0 0 0 0
1 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 1 1 1 1 1 0     0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0     0 0 0 0 0 0 0 1     1 1 0 0 0 0 0 0     0 0 0 1 1 0 0 0
1 0 0 0 0 0 0 0     0 0 0 1 1 1 0 0     0 0 0 0 0 0 0 1     1 1 0 1 0 0 0 0
1                   0                   0                   0
```

```
56
0 0 1 0 0 0 0 0
0 0 0 0 1 0 1 1
0 1 1 0 1 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0

57
1 1 0 0 0 0 0 0
0 0 0 0 0 1 1 1
0 0 1 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0
0

58
0 0 0 0 0 0 0 0
1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0

59
0 1 0 0 0 0 0 1
0 1 0 0 1 1 0 1
0 1 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 1 1 0 0 0 0 1
0 1 0 1 1 1 1 0
0 1 0 0 1 1 0 0
0 1 0 0 0 0 0 0
1

60
0 0 1 0 0 0 0 0
0 0 1 1 1 0 0 0
0 0 0 0 0 1 1 1
0 0 0 0 0 0 0 0
0 1 1 0 1 1 0 0
0 1 0 0 0 1 0 0
0 1 0 0 0 0 0 0
0 1 0 0 0 0 0 0
1

61
0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0 1 1 1 0 1 0 0
0 0 0 0 1 0 0 1
1 1 0 0 0 0 0 0
0 0 1 1 0 1 1 0
0 0 0 0 0 0 1 1
0

62
0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
1 1 0 0 0 0 0 0
0 0 0 1 1 1 0 0
0 0 0 0 0 0 1 1
1

63
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0

64
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1
0 1 1 0 1 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 1 0 1 1
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0

65
0 0 0 0 0 1 1 1
0 0 1 1 1 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1
1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0

66
1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 1 1 1 1 1 0
1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0

67
0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0

68
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0

69
1 0 1 0 0 0 0 0
0 0 0 1 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
1 0 0 1 0 0 0 0
1 0 0 1 0 0 1 0
1

70
0 0 1 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 1 0 0 0 0 0
0 1 1 0 0 0 0 0
0 0 1 0 0 0 0 0
1

71
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 1 0 1 0 0 0 0
0

72
0 1 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 1 1 0 0 0 0 0
1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 1 0 0 0 0 0 0

73
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1

74
0 1 1 0 0 0 0 0
0 0 0 1 1 1 0 0
0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

75
0 1 1 1 1 0 0 0
0 0 0 0 0 1 1 1
1 0 0 0 0 0 0 0
0 1 1 1 0 0 0 0
0 0 0 0 1 1 1 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

```
0 1 0 0 0 0 0 0
1

0 0 0 0 0 0 0 0
0

0 0 0 0 0 0 0 0
0

0 0 0 0 0 0 0 0
0

76
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 1 0 1 0 0 0 0
0 0 0 0 1 1 0 1
0 0 0 0 0 0 0 1
1 1 1 0 0 0 0 0
0 0 0 1 1 1 1 0
0 0 0 0 0 0 0 1
0

77
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 1 1 1 0 0 0 0
0 0 0 0 1 1 1 0
0 0 0 0 0 0 0 1
1 1 1 0 0 0 0 0
0

78
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 1 1 0 0 0 0 0
0 0 0 1 1 1 1 0
0

79
1 0 0 1 0 1 0 1
1 0 0 1 0 0 0 1
1 0 0 1 0 1 0 0
1 0 0 1 0 1 0 0
1 0 0 1 0 1 0 0
1 0 0 0 0 1 0 0
1 0 0 1 0 1 0 0
1 0 0 1 0 1 0 0
1

80
0 1 0 0 0 0 0 0
0 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 1 0 0 0 0 0
0 1 1 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 1 0 0 0 0 0
1

81
0 0 0 1 0 0 0 0
0 1 0 1 0 0 0 0
0 1 0 1 0 0 0 0
0 1 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 1 0 0 0 0 0 0
1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1

82
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0

83
0 0 0 1 1 1 1 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0

84
0 0 0 0 0 0 1 0
1 1 0 0 0 0 0 0
0 1 1 1 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0

85
1 0 0 1 0 1 0 0
1 0 0 1 0 0 0 0
1 0 0 1 0 0 1 0
1 0 0 1 0 0 1 0
1 0 0 1 0 1 0 0
1 0 0 1 0 1 0 0
1 0 1 1 0 1 0 0
0 0 0 0 0 0 0 0
1

86
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0

87
0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1
0 1 1 1 1 1 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0

88
1 1 0 0 0 0 0 0
1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
1 0 1 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1

89
1 0 0 0 0 1 0 0
1 0 0 0 1 0 0 0
1 0 0 0 1 0 0 0
1 0 0 0 1 0 0 0
1 0 0 0 1 0 0 0
1 0 0 0 1 0 0 0
1 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0
1

90
0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0

91
0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 1
0 0 0 0 1 0 0 1
0 0 0 0 1 0 0 1
0 0 0 0 1 0 0 1
0 0 0 0 1 0 0 1
0 0 0 0 1 0 0 1
0 0 0 1 1 0 0 1
1

92
0 0 0 0 0 0 0 0
1 1 0 1 1 1 0 0
0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 1

93
0 0 0 0 1 0 0 1
0 0 0 1 0 0 0 1
0 0 0 1 0 0 0 1
0 0 1 1 0 0 0 1
0 0 1 1 0 0 1 1

94
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 1 1 1 1 1 0 0
1 0 0 0 0 1 0 1
0 0 0 0 0 0 0 0

95
1 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 0 1 1 1
```

```
0 0 0 0 1 0 0 1
0 0 0 0 1 0 0 1
0 0 0 0 1 0 1 0
1

0 0 0 1 0 0 0 1
0 0 0 1 0 0 1 0
0 0 1 1 0 0 0 1
1

0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0

1 0 0 0 1 1 0 0
0 0 0 0 1 0 0 0
1 0 0 1 0 0 0 0
1

96
0 0 0 0 1 0 1 0
0 0 0 0 1 0 0 0
0 0 0 0 1 0 1 0
0 0 0 0 1 0 1 0
0 0 0 0 1 0 1 0
0 0 0 0 1 0 1 0
0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 1
1

97
0 0 0 1 0 0 1 0
0 0 0 1 0 0 1 0
0 0 0 1 0 0 1 0
0 0 0 1 0 0 1 0
0 0 0 1 0 0 1 0
0 0 0 1 0 0 1 0
0 0 0 1 0 0 1 0
0 0 0 1 0 0 1 0
1

98
0 0 0 0 0 0 0 1
0 0 0 0 1 0 0 1
0 0 0 1 1 0 0 1
0 0 0 1 0 0 0 0
1 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
1

99
0 0 0 0 0 1 0 0
0 0 0 1 1 0 0 0
1 0 1 1 0 0 0 0
1 0 1 0 1 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0
1

100
0 0 0 0 0 0 1 0
0 0 0 1 0 0 1 0
0 0 0 1 0 0 1 0
0 0 0 1 0 0 1 0
0 0 0 1 0 0 1 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 1 0
0 0 0 0 0 0 1 0
1

101
0 0 0 0 0 0 1 0
0 0 0 0 0 1 0 0
0 0 0 0 1 1 0 0
0 0 0 0 0 0 1 0
0 1 0 0 0 0 1 0
1 0 0 1 0 0 1 0
0 0 1 1 0 0 1 0
0 0 0 1 0 0 0 0
1

102
0 0 1 0 0 1 0 0
0 1 1 0 0 0 0 1
0 1 0 0 0 1 0 1
1 0 0 0 0 1 0 1
0 0 0 0 0 1 0 1
0 1 1 0 0 1 0 0
0 0 0 1 0 0 0 0
0 0 1 0 0 1 1 0
1

103
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0
1 0 0 1 0 1 0 0
0 0 0 1 0 0 0 0
1 0 0 1 0 0 0 0
1 0 0 1 0 0 0 0
1

104
0 0 0 0 1 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
1 1 0 1 0 0 0 0
0 0 0 0 0 0 0 0
1 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
1

105
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 1 0
1

106
0 0 0 1 0 0 0 1
0 0 0 1 0 1 1 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 1 0 0 0 0 0
1

107
0 0 0 1 0 0 0 1
1 0 0 1 0 0 0 0
1 0 0 1 0 0 0 0
0 0 0 1 0 1 0 0
0 0 0 1 0 1 0 0
0 0 0 1 0 1 0 0
0 0 0 1 0 1 1 0
0 0 0 1 0 0 1 0
1

108
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 1 0 0 0 0 0
1

109
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 1 1 0 0
0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0
1

110
0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0

111
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
1

112
0 0 0 1 0 0 1 0
0 0 0 1 0 0 1 0
0 0 0 1 0 0 0 0

113
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 1 0 0 0 0 0

114
1 0 0 1 1 0 0 0
0 0 0 1 0 0 0 0
0 0 1 0 0 0 0 0

115
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

```
0 0 0 1 0 1 1 0      0 0 1 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 1 0 0 0 0 0
0 0 0 1 0 1 0 0      0 0 1 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0      0 0 1 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 1 1 0 0
0 0 0 1 0 0 1 0      0 1 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0      1 1 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
1                    1                    0                    0

116                  117                  118                  119
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 1 0      1 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 1 0      0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 1 0      0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 1 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 1 0      0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 1 0      0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 1 0      0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 1 0 0      0 0 0 0 0 0 0 0
1                    0                    1                    0

120                  121                  122                  123
0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0      0 0 1 0 0 0 0 0      0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0      0 0 1 0 0 0 0 0      1 0 1 1 1 1 0 0
0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0      0 0 1 0 0 0 0 0      0 0 0 0 0 0 1 1
0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0      0 0 1 0 0 0 0 0      0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0      0 0 1 0 0 0 0 1      0 1 0 0 0 0 0 0
1 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0      0 0 1 0 1 0 0 0      0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0      0 0 1 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
1                    0                    1                    0

124                  125                  126                  127
0 0 0 0 0 1 0 0      0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0      0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
1 1 1 0 1 1 1 1      0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 1 1 0 1 0 0      0 1 0 1 1 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 1 0 0 0 1 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0      0 0 0 0 1 0 1 0      1 1 1 0 0 0 0 0      0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 1 1 1 1      0 0 0 0 0 0 0 0
0                    1                    0                    0

128                  129                  130                  131
0 0 0 0 1 1 1 0      0 0 0 0 0 0 0 0      0 1 1 1 0 1 1 0      0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 1      1 1 1 1 0 0 0 0      0 0 0 0 0 0 0 1      1 1 1 1 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 1 1 1 1      1 1 0 0 0 0 0 0      0 0 0 0 1 1 1 1
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 1 1 1 1 1 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      1 1 1 1 1 0 0 0
0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 1 1 1
0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0                    0                    0                    0

132                  133                  134                  135
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0
1 1 1 0 0 0 0 0
0 0 0 1 1 1 1 1
0 0 0 0 0 0 0 0
1 1 1 0 0 0 0 0
0 0 0 1 0 1 1 1
0 0 0 0 0 0 0 0
0

0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 1 1 0 0 1 1 0
0 0 0 1 1 1 1 1
0 0 0 0 0 0 0 0
1 1 0 0 0 0 0 0
0 0 1 1 1 1 0 0
0

0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1
0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0

0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 0
0 0 0 0 0 0 1 0
0

136
0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0

137
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
1

138
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0

139
0 1 1 1 1 1 0 0
0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0

140
0 0 0 0 0 0 0 0
1 0 1 0 0 0 0 0
0 0 0 1 0 1 1 0
0 0 0 0 1 1 1 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0

141
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 1 0 0 0 0 0
0 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0
1

142
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 1 0 0 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0

143
0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 1 1 0 0 0 0
0 0 0 0 0 0 0 0
1

144
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0

145
0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 0 1 1 1 0 1 1
0 0 0 0 0 1 1 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0

146
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 1 1 1 0 1 1 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0

147
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 1 0 0
0 0 0 0 0 1 0 0
0

148
0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1
0 0 1 1 0 0 0 0
0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 1 1 0 0 0 0
0 0 0 1 1 0 0 0
0 0 0 0 0 1 1 1
0

149
0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1
0

150
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1
0 0 0 0 1 1 0 0
1 1 0 0 1 0 0 0
0 0 0 0 0 0 0 0
0

151
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0
```

```
152                 153                 154                 155
0 0 0 0 0 0 0 0     1 0 1 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0     1 0 1 0 0 1 1 1     0 1 1 0 0 1 1 0     0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 0     1 0 1 0 0 0 0 1     0 0 0 0 0 0 0 1     1 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0     0 0 1 0 0 0 0 0     0 0 1 0 1 0 0 0     0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0     0 0 1 0 0 0 0 1     0 0 0 0 0 0 0 0     1 0 1 0 0 0 0 0
1 0 0 0 0 0 0 0     0 0 1 0 0 0 0 1     0 0 0 0 0 0 0 0     0 0 0 0 0 0 1 0
1 0 0 0 0 0 0 0     0 0 0 0 0 0 0 1     0 0 0 0 0 0 0 0     0 0 0 0 0 1 1 0
0 0 0 0 0 0 1 0     0 0 0 0 0 0 0 1     0 0 0 0 0 0 0 0     0 0 0 0 0 1 0 0
0                   1                   0                   0

156                 157                 158                 159
0 0 0 0 0 1 0 1     0 0 0 0 0 0 0 0     0 0 0 0 0 0 1 0     1 0 0 0 0 1 0 1
0 0 0 0 0 1 0 1     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 1 0 1
0 0 0 0 0 1 0 1     1 0 0 0 0 0 0 0     0 0 0 0 0 1 0 0     0 0 0 0 0 1 0 1
0 0 0 0 0 1 0 1     1 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 1 0 1
0 0 0 0 0 1 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 1 0     0 0 0 0 0 1 0 1
0 0 0 0 0 1 0 1     0 0 0 0 0 0 0 0     0 0 0 0 0 0 1 0     1 0 0 0 0 1 0 1
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 1 0 0     1 0 0 0 0 1 0 1
0 0 0 0 0 1 0 1     0 0 0 0 0 0 0 0     0 0 0 0 1 1 0 0     0 0 0 0 0 0 0 1
1                   0                   0                   1

160                 161                 162                 163
0 1 0 0 0 0 0 0     0 0 0 0 0 1 0 1     0 0 0 0 0 0 1 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 1 0 1     0 0 0 0 0 0 1 0     0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0     0 0 0 0 0 1 0 1     0 0 0 0 0 0 1 0     0 0 0 0 0 1 1 0
0 0 0 0 0 0 0 0     0 0 0 0 0 1 0 1     0 0 0 0 0 1 1 0     0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 1 0 1     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 1 0 0     1 0 0 0 0 1 0 0     0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 1 0 1     0 0 0 0 0 0 0 0     0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 1 0 0 1     0 0 0 0 0 1 0 0     0 0 0 0 0 1 1 0
1                   1                   1                   0

164                 165                 166                 167
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 1 0     0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 1 0     0 0 0 1 1 1 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     1 1 0 1 0 0 1 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 1 0 1 1 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 1 0     1 1 0 0 0 0 0 0     0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0     0 0 0 1 1 1 1 0     1 1 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1     0 1 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 1 1 0
0 0 0 0 1 1 1 1     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 1 0 0
0                   0                   0                   0

168                 169                 170                 171
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 1 1
0 1 1 1 0 0 0 0     1 1 1 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 1
0 0 0 1 0 0 0 0     0 0 0 1 1 1 1 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 1     0 0 0 0 0 0 0 0     0 0 0 0 1 0 0 1
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 1 0 1
0 0 0 0 0 0 0 0     0 0 0 0 0 0 1 1     1 1 0 0 0 0 0 0     0 0 0 0 1 0 0 1
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 1 1 1 1 0 0     0 0 0 0 1 0 0 1
```

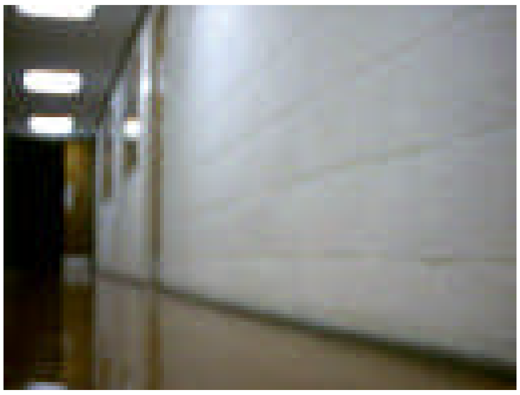
```
0                   0                   0                   1

172                 173                 174                 175
0 0 0 0 0 0 0 0     0 0 0 0 0 0 1 1     1 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     1 1 0 0 0 0 0 0     0 1 1 1 1 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 1 1 0 0 0     0 0 0 0 0 1 1 1     1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 1 1 1     1 0 0 0 0 0 0 0     0 0 1 1 1 1 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 1 1 1 1 0 0 0     0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 1 1 1     1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 1 1 1 1 0 0 0
0 0 0 0 0 0 1 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 1 1 1
0                   0                   0                   0

176                 177                 178                 179
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 1 1 1 0 0 0     0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 1 0     1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 1 1 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 1 1 0
1 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 1
0 1 1 1 1 1 1 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1     1 1 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
1 1 0 0 0 0 0 0     0 0 1 1 1 1 1 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0                   0                   0                   0

180                 181                 182                 183
1 1 1 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 1 0 0
0 0 0 1 1 1 1 0     1 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 1 1 0 0
0 0 0 0 0 0 0 0     1 1 1 1 1 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 1 1 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
1 0 1 1 0 0 0 0     0 0 0 0 0 0 1 0     0 0 0 0 0 0 0 0     0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 1     1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     1 0 1 0 0 0 0 0     0 0 0 1 1 1 1 0     1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0                   0                   0                   0

184                 185                 186                 187
0 1 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0     0 0 0 1 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0     0 0 0 1 0 1 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     1 1 1 1 0 0 0 0     0 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 1 0 0 0 0     0 0 0 0 1 1 1 1     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 1 0 1 0 0     0 0 0 0 0 0 0 0     1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0     0 0 0 1 0 0 0 0     1 1 1 0 0 0 0 0     0 0 0 0 0 1 1 1
0 0 0 0 0 0 0 0     0 0 0 1 0 0 0 1     0 0 0 0 1 1 1 0     0 0 0 0 0 0 0 0
0                   1                   0                   0

188                 189                 190                 191
0 0 0 0 0 0 0 0     1 0 1 0 0 0 0 0     0 0 0 0 0 0 1 1     1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 1 0 1 1 1     0 0 0 0 0 0 0 0     0 1 1 1 1 1 1 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     1 1 1 0 0 0 0 0     0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 1 1 0 1 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 1 0     1 0 1 1 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
```

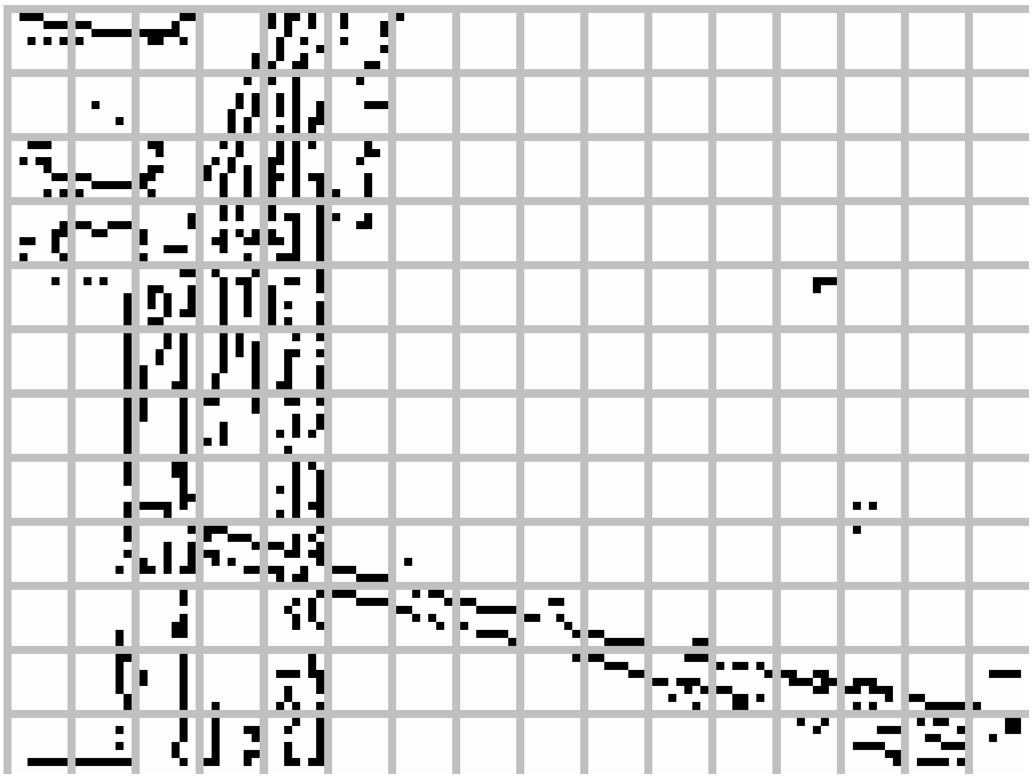
```
1 0 0 0 0 0 0 0
0 1 1 1 1 0 0 0
0

0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0

0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0

0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 0
0

192
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 0 1 0 0 0 0 0
0 0 1 0 1 0 1 1
0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0
0

193
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
1 1 1 1 1 1 1 0
0 0 1 1 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0

194
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 1 1 1 1 1 1 0
0 0 0 0 0 0 0 0
0

195
0 0 0 0 1 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0
0

196
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 1 1 0 0 0
0 0 0 0 0 0 0 0
0

197
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0
0 1 0 1 0 0 0 0
0 1 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0

198
0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 1 0 0
0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 1 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 1
1

199
0 1 0 0 1 0 0 0
0 1 0 0 1 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0
0

200
0 0 0 0 0 0 0 0
1 0 0 1 1 0 0 0
0 1 0 0 1 0 0 0
0 1 0 0 0 1 0 0
0 0 1 0 0 0 1 0
0 0 1 0 0 0 1 0
0 0 0 1 0 0 0 1
0 0 0 1 1 0 0 0
0

201
0 0 0 0 0 0 1 1
0 0 0 0 0 1 1 0
0 0 0 0 1 0 0 1
0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0
1 0 0 0 1 1 0 0
1 0 0 0 0 1 1 1
0 1 0 0 0 0 0 0
0

202
0 1 1 0 0 0 1 1
0 0 0 0 1 1 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 0 1 1 1 1 1 1
1 0 0 0 0 0 0 0
0

203
0 1 0 0 0 0 0 0
0 1 1 1 1 0 0 1
0 0 0 0 0 1 0 0
1 0 1 0 0 0 0 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0

204
0 0 1 0 0 1 1 0
0 0 0 1 0 1 0 0
0 0 0 1 0 1 0 0
0 0 0 1 0 1 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
1

205
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 1 1 0 0 1 1 1
1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0
0

206
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 1
1 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0
0

207
0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1
1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1
1 0 1 1 1 0 0 0
1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0

208
0 0 0 1 1 1 1 1
1 1 1 0 0 0 0 0
1 0 0 0 0 0 0 0
0 1 1 1 1 0 0 1

209
0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 0
0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 0

210
0 0 0 0 0 1 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 1 0 1

211
0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 1 0 0
```

```
1 1 0 0 0 0 0 0     0 0 1 0 0 0 0 0     0 0 0 0 0 1 0 1     0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0     0 0 0 1 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0     0 0 1 0 0 0 0 0     0 0 0 0 0 1 0 0     0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0     0 0 1 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0                   1                   1                   1

212                 213                 214                 215
0 0 0 0 1 0 0 0     0 0 1 0 0 0 0 1     0 0 0 0 1 0 0 1     1 0 0 0 0 0 0 1
0 0 0 0 1 0 0 0     0 0 1 0 0 0 0 1     0 0 0 0 1 0 0 1     0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0     0 0 0 0 0 0 0 1     0 0 0 0 1 0 0 1     0 0 0 0 0 0 0 1
0 0 0 0 1 0 0 0     0 0 0 0 0 0 0 1     0 0 0 0 0 0 0 1     0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0     0 0 0 0 0 0 0 1     0 0 0 0 1 0 0 1     0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 1     0 0 0 0 1 0 0 1     1 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 1     0 0 0 0 1 0 0 1     1 0 1 0 0 0 0 1
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 1 0 0 0 0 1
1                   1                   1                   1

216                 217                 218                 219
1 0 0 0 0 0 0 1     0 1 0 0 0 0 0 0     0 0 1 0 0 0 0 0     0 0 0 1 0 0 0 0
1 0 0 0 0 0 0 0     0 1 0 0 0 0 0 0     0 0 1 0 0 0 0 0     0 0 0 1 0 0 0 0
1 0 0 0 0 0 0 0     0 1 0 0 0 0 0 0     0 0 1 0 0 0 0 0     0 0 0 1 0 0 0 0
1 0 0 0 0 0 0 0     0 1 0 0 0 0 0 0     0 0 1 0 0 0 0 0     0 0 0 1 0 0 0 0
1 0 0 0 0 0 0 0     0 1 0 0 0 0 0 0     0 0 1 0 0 0 0 0     0 0 0 1 0 0 0 0
1 0 1 0 0 0 0 0     0 1 0 0 0 0 0 0     0 0 1 0 0 0 0 0     0 0 0 1 0 0 0 0
1 0 0 0 0 0 0 0     0 1 0 0 0 0 0 0     0 0 1 0 0 0 0 0     0 0 0 1 0 0 0 0
1 0 0 0 0 0 0 1     0 1 0 0 0 0 0 0     0 0 1 0 0 0 0 0     0 0 0 1 0 0 0 0
1                   1                   1                   1

220                 221                 222                 223
0 0 0 0 1 0 0 0     0 0 0 0 0 1 0 0     0 0 0 0 0 0 1 0     0 0 0 0 0 0 0 1
0 0 0 0 1 0 0 0     0 0 0 0 0 1 0 0     0 0 0 0 0 0 1 0     0 0 0 0 0 0 0 1
0 0 0 0 1 0 0 0     0 0 0 0 0 1 0 0     0 0 0 0 0 0 1 0     0 0 0 0 0 0 0 1
0 0 0 0 1 0 0 0     0 0 0 0 0 1 0 0     0 0 0 0 0 0 1 0     0 0 0 0 0 0 0 1
0 0 0 0 1 0 0 0     0 0 0 0 0 1 0 0     0 0 0 0 0 0 1 0     0 0 0 0 0 0 0 1
0 0 0 0 1 0 0 0     0 0 0 0 0 1 0 0     0 0 0 0 0 0 1 0     0 0 0 0 0 0 0 1
0 0 0 0 1 0 0 0     0 0 0 0 0 1 0 0     0 0 0 0 0 0 1 0     0 0 0 0 0 0 0 1
0 0 0 0 1 0 0 0     0 0 0 0 0 1 0 0     0 0 0 0 0 0 1 0     0 0 0 0 0 0 0 1
1                   1                   1                   1
```

Sample Test Results

The input image is shown broken down into sub-images. Note that lines of the patition have covered some data.





There were 16 mis-recognitions in this test. The include sub images 4, 50, 65, 99, 113, 130, 141, 146, 148, 161, 162, 170, 173, 180, 189 and 190.

```
0
0 0 0 0 0 0 0 0
0 0 1 1 1 0 0 0
0 0 0 0 0 1 1 1
0 0 0 0 0 0 0 0
0 0 0 1 0 1 0 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

Sub-image ID: 0
Back propagation network result: 0.0855453
Back propagation network result to nearest integer: 0
Expected: 0

1

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 1 1 0 0 0 0 0
0 0 0 1 1 1 1 1
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 1
Back propagation network result: 0.00310737
Back propagation network result to nearest integer: 0
Expected: 0

2
0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1
0 0 0 0 0 1 0 0
1 1 1 1 1 1 0 0
0 0 1 1 0 0 1 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 2
Back propagation network result: 0.000664517
Back propagation network result to nearest integer: 0
Expected: 0

3
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1

Sub-image ID: 3
Back propagation network result: 0.00398829
Back propagation network result to nearest integer: 0
Expected: 0

4
0 0 0 0 0 0 0 0
0 1 0 1 1 0 1 0
0 1 0 1 0 0 1 0
0 0 0 1 0 0 0 0
1 0 1 0 0 1 0 0
0 0 1 0 0 0 0 1
0 0 1 0 0 1 0 0
0 1 0 0 1 1 0 0
```

```
Sub-image ID: 4
Back propagation network result: 0.31501
Back propagation network result to nearest integer: 0
Expected: 0

5
0 0 0 0 0 0 0 0
1 0 1 0 0 0 0 0
0 0 1 0 0 0 0 1
1 0 0 0 0 0 0 1
1 0 1 0 0 0 0 0
1 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0
0 0 0 0 0 1 1 0

Sub-image ID: 5
Back propagation network result: 0.984435
Back propagation network result to nearest integer: 1
Expected: 1

6
0 0 0 0 0 0 0 0
1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0

Sub-image ID: 6
Back propagation network result: 0.00619991
Back propagation network result to nearest integer: 0
Expected: 0

7
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 7
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

8
0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 8
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

9
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 9
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

10
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 10
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

11
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

```
Sub-image ID: 11
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

12
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 12
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

13
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 13
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

14
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 14
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

15
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 15
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

16
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 16
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

17
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0

Sub-image ID: 17
Back propagation network result: 0.000467102
Back propagation network result to nearest integer: 0
Expected: 0

18
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 18
```

```
Back propagation network result: 0.000333478
Back propagation network result to nearest integer: 0
Expected: 0

19
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 1
0 0 0 0 0 1 0 1
0 0 0 0 1 0 0 1
0 0 0 0 1 0 1 0
0 0 0 0 1 0 1 0

Sub-image ID: 19
Back propagation network result: 0.125819
Back propagation network result to nearest integer: 0
Expected: 0

20
0 1 0 0 1 0 0 0
1 1 0 0 1 0 0 0
1 0 0 0 1 0 0 0
0 0 1 0 1 0 0 0
0 0 1 0 1 0 0 1
0 0 1 0 1 0 0 1
0 0 0 0 1 0 1 1
0 0 1 0 1 0 1 0

Sub-image ID: 20
Back propagation network result: 0.99976
Back propagation network result to nearest integer: 1
Expected: 1

21
1 0 0 0 0 0 0 0
1 0 0 0 1 0 0 0
1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 21
Back propagation network result: 0.226474
Back propagation network result to nearest integer: 0
Expected: 0

22
0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

```
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 22
Back propagation network result: 0.0035224
Back propagation network result to nearest integer: 0
Expected: 0

23
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 23
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

24
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 24
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

25
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 25
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

26
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 26
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

27
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 27
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

28
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 28
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

29
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 29
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

30
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 30
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

31
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 31
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

32
0 0 0 0 0 1 1 1
0 0 0 1 1 1 0 0
0 0 0 0 0 0 0 0
0 0 1 0 1 1 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 1

Sub-image ID: 32
Back propagation network result: 0.995001
Back propagation network result to nearest integer: 1
```

```
Expected: 1

33
1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 1 1 0 0 0 0 0
0 0 0 1 1 1 1 1
0 1 0 0 0 0 0 0

Sub-image ID: 33
Back propagation network result: 0.244134
Back propagation network result to nearest integer: 0
Expected: 0

34
1 1 1 0 0 0 0 0
0 0 1 1 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 0
0 1 1 0 0 0 0 0
1 1 0 0 0 0 0 0
0 0 1 0 0 0 0 0

Sub-image ID: 34
Back propagation network result: 0.999066
Back propagation network result to nearest integer: 1
Expected: 1

35
0 0 0 0 0 1 0 0
0 0 0 1 0 1 0 0
0 0 0 0 0 1 0 0
0 0 1 0 1 0 0 0
0 1 0 0 1 0 1 0
0 1 0 1 0 0 1 0
0 0 0 1 0 0 1 0
1 0 0 1 0 0 1 0

Sub-image ID: 35
Back propagation network result: 0.99645
Back propagation network result to nearest integer: 1
Expected: 1

36
0 0 0 0 1 0 1 0
0 0 1 0 1 0 1 0
0 0 1 0 1 0 0 0
0 1 1 0 1 0 0 0
0 1 0 0 1 0 0 0
```

```
0 1 1 0 1 0 1 1
0 1 0 0 1 0 0 1
1 1 0 0 1 0 0 1

Sub-image ID: 36
Back propagation network result: 0.999185
Back propagation network result to nearest integer: 1
Expected: 1

37
0 0 0 0 0 1 0 1
0 0 0 0 0 1 0 0
0 0 0 0 0 1 1 0
1 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 1 0 0
0 1 0 0 0 1 0 0

Sub-image ID: 37
Back propagation network result: 0.965308
Back propagation network result to nearest integer: 1
Expected: 1

38
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 38
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

39
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 39
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0
```

```
40
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 40
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

41
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 41
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

42
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 42
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

43
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 43
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

44
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 44
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

45
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 45
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

46
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 46
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0
```

```
47
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 47
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

48
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 1 1
0 0 1 1 0 0 1 0
0 0 0 0 0 0 1 0
0 0 1 0 0 0 0 1

Sub-image ID: 48
Back propagation network result: 0.458966
Back propagation network result to nearest integer: 0
Expected: 0

49
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 1 1 0 0 1 1 1
0 0 0 1 1 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 49
Back propagation network result: 0.0102242
Back propagation network result to nearest integer: 0
Expected: 0

50
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 1 1 1 0
```

```
1 1 0 0 0 0 0 0

Sub-image ID: 50
Back propagation network result: 0.743732
Back propagation network result to nearest integer: 1
Expected: 1

51
0 0 1 0 0 1 0 0
0 0 0 1 0 1 0 0
0 0 0 1 0 1 0 0
0 0 0 0 0 0 0 0
0 0 0 1 0 1 0 1
1 0 1 1 0 0 1 1
0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 0

Sub-image ID: 51
Back propagation network result: 0.999469
Back propagation network result to nearest integer: 1
Expected: 1

52
0 1 0 0 1 0 0 1
0 1 0 0 0 0 0 1
0 1 0 1 1 0 0 1
0 0 0 0 1 0 0 1
1 1 0 0 1 0 0 1
0 1 1 0 1 0 0 1
0 0 0 0 1 0 0 1
0 0 1 1 1 0 0 1

Sub-image ID: 52
Back propagation network result: 0.999828
Back propagation network result to nearest integer: 1
Expected: 1

53
0 1 0 0 0 1 0 0
0 0 0 0 0 0 0 0
0 1 0 0 0 1 0 0
0 0 0 0 1 1 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 53
Back propagation network result: 0.582711
Back propagation network result to nearest integer: 1
Expected: 1

54
```

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 54
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

55
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 55
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

56
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 56
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

57
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

```
Sub-image ID: 57
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

58
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 58
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

59
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 59
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

60
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 60
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

61
0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 61
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

62
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 62
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

63
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 63
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

64
0 1 0 1 1 1 0 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

```
Sub-image ID: 64
Back propagation network result: 0.3123
Back propagation network result to nearest integer: 0
Expected: 0

65
1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0
1 0 1 0 1 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1

Sub-image ID: 65
Back propagation network result: 0.994667
Back propagation network result to nearest integer: 1
Expected: 1

66
1 0 1 1 1 1 1 0
0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 1
0 0 1 0 1 0 0 1
0 0 1 0 1 0 0 1
0 0 0 0 1 0 1 1
0 0 1 1 0 0 0 0

Sub-image ID: 66
Back propagation network result: 0.977365
Back propagation network result to nearest integer: 1
Expected: 1

67
0 0 1 0 0 0 0 1
0 0 1 0 0 0 0 1
0 0 0 1 0 1 1 0
0 0 0 1 0 0 1 0
0 0 0 1 0 0 1 0
0 0 0 1 0 0 1 0
0 0 0 1 0 0 1 0
0 0 0 1 0 0 0 0

Sub-image ID: 67
Back propagation network result: 0.906462
Back propagation network result to nearest integer: 1
Expected: 1

68
0 1 0 0 1 0 0 1
1 0 0 0 0 0 0 1
```

```
0 0 0 1 1 0 0 1
0 1 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 1 0 1 0 0 0 1
0 1 0 0 0 0 0 1
0 1 0 1 0 0 0 1

Sub-image ID: 68
Back propagation network result: 0.999633
Back propagation network result to nearest integer: 1
Expected: 1

69
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 69
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

70
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 70
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

71
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 71
```

```
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

72
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 72
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

73
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 73
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

74
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 74
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

75
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 75
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

76
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1
0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 76
Back propagation network result: 0.00179131
Back propagation network result to nearest integer: 0
Expected: 0

77
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 77
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

78
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 78
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

79
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 79
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

80
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 80
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

81
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1

Sub-image ID: 81
Back propagation network result: 0.965289
Back propagation network result to nearest integer: 1
Expected: 1

82
0 0 0 0 0 0 1 0
0 0 0 0 1 0 1 0
0 0 0 0 1 0 1 0
0 0 0 1 0 0 1 0
```

```
0 0 0 1 0 0 1 0
0 1 0 0 0 0 1 0
0 1 0 0 0 0 1 0
0 1 0 0 0 1 1 0

Sub-image ID: 82
Back propagation network result: 0.974531
Back propagation network result to nearest integer: 1
Expected: 1

83
0 0 0 1 0 1 0 0
0 0 0 1 0 1 0 0
0 0 0 1 0 1 0 1
0 0 0 1 0 1 0 1
0 0 0 1 0 0 0 1
0 0 0 1 0 0 0 1
0 0 1 0 0 0 0 1
0 0 1 0 0 0 0 1

Sub-image ID: 83
Back propagation network result: 0.999164
Back propagation network result to nearest integer: 1
Expected: 1

84
1 0 0 0 1 0 0 1
1 0 0 0 1 0 0 1
1 0 0 0 0 0 0 0
0 0 0 1 1 0 0 1
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 1
0 0 0 1 0 0 0 1
0 0 1 1 0 0 0 1

Sub-image ID: 84
Back propagation network result: 0.998935
Back propagation network result to nearest integer: 1
Expected: 1

85
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 85
Back propagation network result: 0.0118129
Back propagation network result to nearest integer: 0
```

```
Expected: 0

86
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 86
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

87
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 87
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

88
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 88
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

89
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 89
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

90
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 90
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

91
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 91
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

92
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 92
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0
```

```
93
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 93
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

94
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 94
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

95
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 95
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

96
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 96
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

97
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1

Sub-image ID: 97
Back propagation network result: 0.965289
Back propagation network result to nearest integer: 1
Expected: 1

98
0 1 0 0 0 1 1 0
0 1 0 0 0 0 1 0
0 1 0 0 0 0 1 0
0 1 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0

Sub-image ID: 98
Back propagation network result: 0.966981
Back propagation network result to nearest integer: 1
Expected: 1

99
0 0 0 0 0 0 0 1
0 1 1 0 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0
0 1 0 1 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 99
Back propagation network result: 0.926362
Back propagation network result to nearest integer: 1
Expected: 1
```

```
100
0 0 0 1 0 0 0 1
0 0 0 1 1 0 0 1
1 0 0 0 0 0 0 0
0 0 0 0 1 0 0 1
0 0 0 0 1 0 0 1
0 0 1 0 1 0 1 0
0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0

Sub-image ID: 100
Back propagation network result: 0.99968
Back propagation network result to nearest integer: 1
Expected: 1

101
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 101
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

102
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 102
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

103
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0

Sub-image ID: 103
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

104
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 104
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

105
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 105
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

106
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 106
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

107
```

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 107
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

108
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 108
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

109
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 109
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

110
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

```
Sub-image ID: 110
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

111
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 111
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

112
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 112
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

113
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1

Sub-image ID: 113
Back propagation network result: 0.0159447
Back propagation network result to nearest integer: 0
Expected: 0

114
0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 1 1 0
0 0 0 0 0 1 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 1
1 1 1 1 1 0 1 0
0 0 0 0 1 0 0 0

Sub-image ID: 114
Back propagation network result: 0.855666
Back propagation network result to nearest integer: 1
Expected: 1

115
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 115
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

116
0 0 0 0 0 0 1 0
0 0 0 0 1 0 1 0
0 0 0 0 1 0 0 1
0 0 0 0 1 0 0 1
0 0 1 0 1 0 0 1
0 0 0 0 1 0 0 1
0 0 0 0 1 0 1 1
0 0 1 0 1 0 0 1

Sub-image ID: 116
Back propagation network result: 0.999397
Back propagation network result to nearest integer: 1
Expected: 1

117
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

```
Sub-image ID: 117
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

118
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 118
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

119
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 119
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

120
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 120
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

121
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 121
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

122
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 122
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

123
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 123
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

124
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 124
```

```
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

125
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 125
Back propagation network result: 0.00156585
Back propagation network result to nearest integer: 0
Expected: 0

126
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 126
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

127
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 127
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

128
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 128
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

129
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0

Sub-image ID: 129
Back propagation network result: 0.0372545
Back propagation network result to nearest integer: 0
Expected: 0

130
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 1
0 0 0 0 1 0 0 1
1 1 0 0 1 0 0 1
0 1 1 0 1 0 1 1
0 0 0 0 0 0 0 0

Sub-image ID: 130
Back propagation network result: 0.141143
Back propagation network result to nearest integer: 0
Expected: 0

131
0 0 0 0 0 0 1 0
1 1 1 1 0 0 0 0
0 1 0 0 1 1 1 0
0 0 0 0 0 0 0 1
0 1 1 0 0 0 0 0
0 0 1 0 1 0 0 0
0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 0

Sub-image ID: 131
Back propagation network result: 0.069524
Back propagation network result to nearest integer: 0
Expected: 0

132
0 0 0 0 1 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 1 0 1 0
1 1 1 0 1 0 0 1
0 0 0 1 1 0 1 1
0 0 0 0 0 0 0 1
1 1 1 0 0 1 0 0
0 0 1 0 1 1 0 0

Sub-image ID: 132
Back propagation network result: 0.932387
Back propagation network result to nearest integer: 1
Expected: 1

133
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0 1 1 1 0 0 0 0
0 0 0 0 1 1 1 1

Sub-image ID: 133
Back propagation network result: 0.0301443
Back propagation network result to nearest integer: 0
Expected: 0

134
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 134
Back propagation network result: 0.0332247
Back propagation network result to nearest integer: 0
Expected: 0

135
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 135
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

136
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 136
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

137
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 137
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

138
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 138
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
```

```
Expected: 0

139
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 139
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

140
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 140
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

141
1 1 0 0 1 1 1 1
0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 141
Back propagation network result: 0.64438
Back propagation network result to nearest integer: 1
Expected: 1

142
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 142
Back propagation network result: 0.000601813
Back propagation network result to nearest integer: 0
Expected: 0

143
1 1 0 1 1 1 1 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 143
Back propagation network result: 0.40142
Back propagation network result to nearest integer: 0
Expected: 0

144
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 144
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

145
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0

Sub-image ID: 145
Back propagation network result: 0.00406566
Back propagation network result to nearest integer: 0
Expected: 0
```

```
146
0 0 0 0 0 0 1 1
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 0 1 1 0
0 0 0 0 0 1 1 0
0 0 0 0 0 0 0 0

Sub-image ID: 146
Back propagation network result: 0.323584
Back propagation network result to nearest integer: 0
Expected: 0

147
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 147
Back propagation network result: 0.0306185
Back propagation network result to nearest integer: 0
Expected: 0

148
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 0 1 0 1 0
0 0 0 1 0 0 1 0
0 0 0 0 1 0 1 0
0 0 0 0 1 0 0 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 148
Back propagation network result: 0.00793975
Back propagation network result to nearest integer: 0
Expected: 0

149
0 0 0 0 0 0 0 0
0 1 1 1 0 0 0 0
0 0 0 0 1 1 1 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 149
Back propagation network result: 0.00272549
Back propagation network result to nearest integer: 0
Expected: 0

150
1 1 1 0 0 0 0 0
0 0 0 1 0 1 1 0
0 0 0 0 0 0 0 1
1 1 1 0 0 0 0 0
0 0 0 1 0 1 0 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 150
Back propagation network result: 0.213158
Back propagation network result to nearest integer: 0
Expected: 0

151
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 1 1 0 0 0 0 0
0 0 0 1 1 1 1 1
0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 0 0 1 1 1 1 0
0 0 0 0 0 0 0 1

Sub-image ID: 151
Back propagation network result: 0.0247189
Back propagation network result to nearest integer: 0
Expected: 0

152
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 1 1 0 0
0 0 0 0 0 0 0 0
1 1 1 0 0 1 0 0
1 0 0 0 0 0 1 0
0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0

Sub-image ID: 152
Back propagation network result: 0.151484
Back propagation network result to nearest integer: 0
Expected: 0
```

```
153
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0 1 1 0 0 0 0 0
0 0 0 1 1 1 1 1

Sub-image ID: 153
Back propagation network result: 0.00721088
Back propagation network result to nearest integer: 0
Expected: 0

154
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1

Sub-image ID: 154
Back propagation network result: 0.0714404
Back propagation network result to nearest integer: 0
Expected: 0

155
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 155
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

156
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0

Sub-image ID: 156
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

157
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 157
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

158
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 158
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

159
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 159
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

160
```

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 160
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

161
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 1
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 1

Sub-image ID: 161
Back propagation network result: 0.315363
Back propagation network result to nearest integer: 0
Expected: 0

162
0 0 0 0 0 0 1 0
1 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 1 0 0 0 0 1 0
0 1 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0

Sub-image ID: 162
Back propagation network result: 0.168813
Back propagation network result to nearest integer: 0
Expected: 0

163
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0
```

```
Sub-image ID: 163
Back propagation network result: 0.000861923
Back propagation network result to nearest integer: 0
Expected: 0

164
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 1 1 1 0 1 1
0 0 0 0 0 0 0 1
0 0 0 1 0 0 0 1
0 0 0 1 0 0 0 0
0 0 1 0 1 0 0 1

Sub-image ID: 164
Back propagation network result: 0.72678
Back propagation network result to nearest integer: 1
Expected: 1

165
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 165
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

166
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 166
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

167
0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 167
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

168
0 0 1 1 0 1 1 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 168
Back propagation network result: 0.000137584
Back propagation network result to nearest integer: 0
Expected: 0

169
0 0 0 0 0 0 0 0
1 1 1 0 0 0 0 0
0 0 0 1 1 1 0 0
0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 169
Back propagation network result: 0.0376559
Back propagation network result to nearest integer: 0
Expected: 0

170
1 1 1 1 1 0 0 0
0 0 0 0 0 1 1 1
0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0
1 1 1 0 1 1 1 0
0 0 0 0 0 1 0 1
0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 0
```

```
Sub-image ID: 170
Back propagation network result: 0.990589
Back propagation network result to nearest integer: 1
Expected: 1

171
0 0 0 0 1 1 0 0
1 0 0 0 0 0 0 0
0 1 0 1 1 0 1 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0
0 1 1 0 0 0 0 0
0 0 0 1 1 0 1 0
1 0 0 1 1 0 0 0

Sub-image ID: 171
Back propagation network result: 0.0947641
Back propagation network result to nearest integer: 0
Expected: 0

172
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 1 1 0 0 1 1 0
0 0 1 1 1 0 1 1
0 0 0 0 0 1 1 0
0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0

Sub-image ID: 172
Back propagation network result: 0.085181
Back propagation network result to nearest integer: 0
Expected: 0

173
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 0 1 1 1 0 0 0
0 1 1 0 1 1 1 0
0 0 0 0 0 0 1 0
0 0 1 0 1 0 0 0

Sub-image ID: 173
Back propagation network result: 0.724906
Back propagation network result to nearest integer: 1
Expected: 1

174
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 1 1 0 0 0 0 0
0 0 0 1 1 1 1 1

Sub-image ID: 174
Back propagation network result: 0.0143276
Back propagation network result to nearest integer: 0
Expected: 0

175
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
1 1 0 0 0 0 0 0

Sub-image ID: 175
Back propagation network result: 0.106286
Back propagation network result to nearest integer: 0
Expected: 0

176
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1
0 0 0 0 0 0 0 0

Sub-image ID: 176
Back propagation network result: 0.00019055
Back propagation network result to nearest integer: 0
Expected: 0

177
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0

Sub-image ID: 177
```

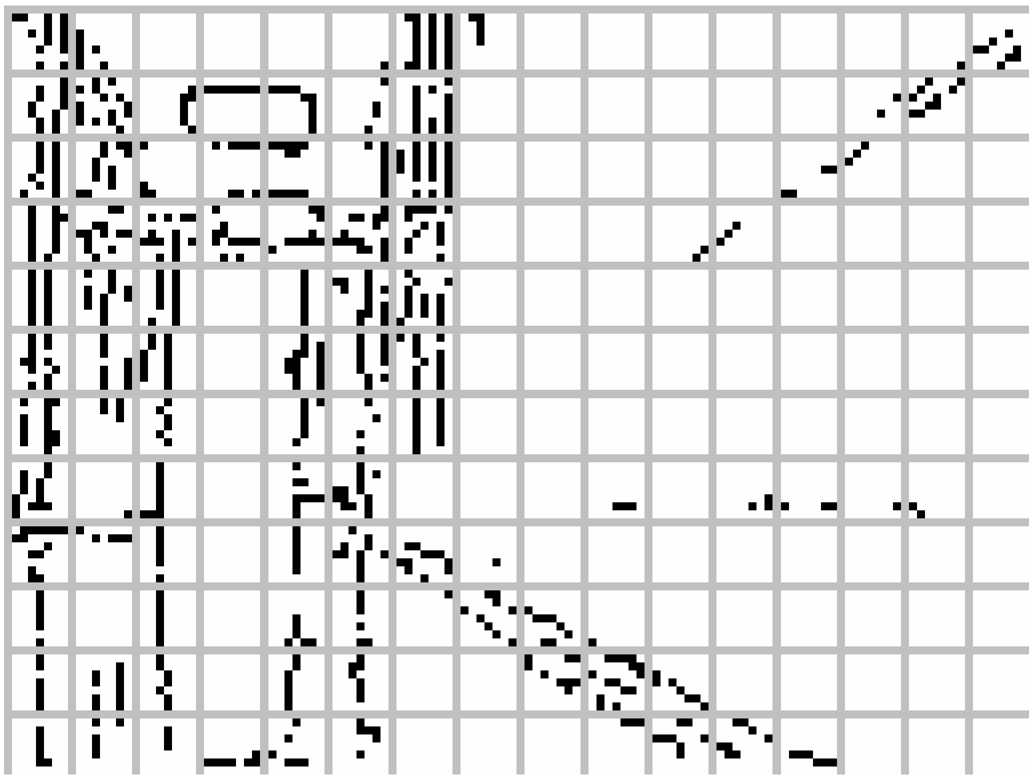
```
Back propagation network result: 0.000245643
Back propagation network result to nearest integer: 0
Expected: 0

178
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0
0 0 0 0 0 1 0 0
0 0 0 0 0 1 0 0
1 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0

Sub-image ID: 178
Back propagation network result: 0.604369
Back propagation network result to nearest integer: 1
Expected: 1

179
1 0 1 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 1 0 0 0 1 1
0 0 1 0 0 0 0 1
0 0 1 0 0 0 0 0
0 0 1 0 0 0 1 1
0 1 1 0 0 0 1 0
0 0 0 0 0 0 0 0

Sub-image ID: 179
Back propagation network result: 0.914853
Back propagation network result to nearest integer: 1
Expected: 1

180
0 0 0 0 1 0 0 1
0 0 0 0 1 0 0 1
0 0 0 1 0 0 0 1
0 0 0 0 0 0 0 1
0 0 0 1 0 0 0 1
0 0 0 1 0 0 0 1
0 0 0 1 1 0 1 1
0 0 0 0 0 0 0 0

Sub-image ID: 180
Back propagation network result: 0.20389
Back propagation network result to nearest integer: 0
Expected: 0

181
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 181
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

182
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 182
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

183
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 183
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

184
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 184
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

185
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 185
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

186
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 186
Back propagation network result: 0.000163529
Back propagation network result to nearest integer: 0
Expected: 0

187
0 0 0 0 0 1 1 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 187
Back propagation network result: 0.00147141
Back propagation network result to nearest integer: 0
Expected: 0

188
0 0 1 0 0 0 0 0
0 0 0 1 0 0 1 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0
```

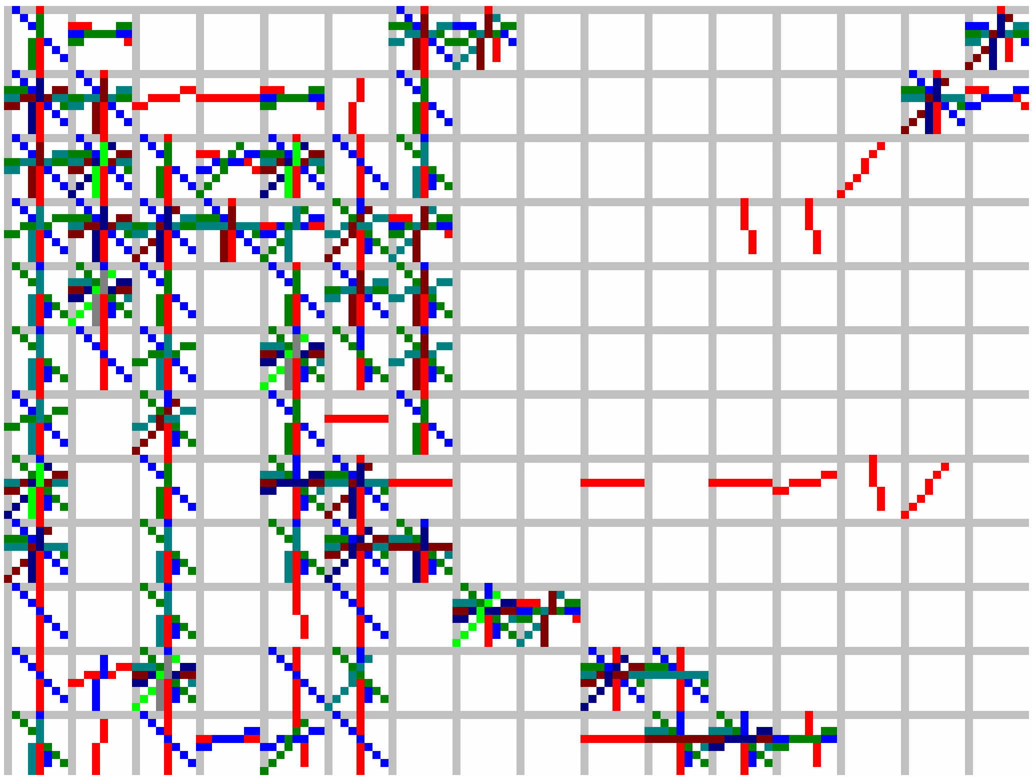
```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

Sub-image ID: 188
Back propagation network result: 0.00467777
Back propagation network result to nearest integer: 0
Expected: 0

189
0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0
1 0 0 0 0 1 1 1
1 0 0 0 0 0 0 0
0 0 1 1 1 1 0 0
0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0

Sub-image ID: 189
Back propagation network result: 0.758459
Back propagation network result to nearest integer: 1
Expected: 1

190
1 0 0 0 0 0 0 0
0 0 1 0 1 1 0 0
1 0 0 0 0 0 0 1
0 0 0 1 1 0 0 0
0 0 0 0 0 1 1 1
0 0 0 0 0 0 0 0
1 0 1 1 1 1 0 1
0 0 0 0 0 0 0 0

Sub-image ID: 190
Back propagation network result: 0.998176
Back propagation network result to nearest integer: 1
Expected: 1

191
0 0 1 1 1 0 0 0
0 0 0 0 0 1 1 0
1 0 0 0 0 1 1 0
0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0


Sub-image ID: 191
Back propagation network result: 0.404942
Back propagation network result to nearest integer: 0
Expected: 0
```

Appendix D Sample Sub-Image Line Recognition Results for a Full Image

Sample Thinned Binary Image



Results from Sample Image



Testing Set

Following are sub-images derived from breaking down the image shown above in numberic format. Each sub-image is preceded by its ID.

```
0                  1                  2                  3
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 1 1 0 0 1 0 1    1 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 1    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 1 0 1 0 1    0 1 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 1    0 1 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 1    0 1 0 1 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 1 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 1    0 1 0 0 1 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0

4                  5                  6                  7
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    1 0 1 1 0 1 0 1    1 0 1 1 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    1 0 0 1 0 1 0 1    0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    1 0 0 1 0 1 0 1    0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    1 0 0 1 0 1 0 1    0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    1 0 0 1 0 1 0 1    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 1 0 1 0 1    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 1    0 0 1 1 0 1 0 1    0 0 0 0 0 0 0 0

8                  9                  10                 11
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0

12                 13                 14                 15
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 1 1 0 0 0 1 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    1 0 0 0 0 1 1 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 1    0 0 0 0 1 0 0 0

16                 17                 18                 19
0 0 0 0 1 0 0 1    0 0 1 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1    0 0 0 1 0 1 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 1    0 1 0 1 0 0 0 0    0 0 0 0 0 0 0 1    1 1 1 1 1 1 1 1
0 0 0 0 1 0 0 1    0 0 0 0 1 0 1 0    0 0 0 0 0 0 1 1    0 0 0 0 0 0 0 0
```

```
0 0 0 1 0 0 0 1      0 1 0 0 0 0 0 1      0 0 0 0 0 0 1 0      0 0 0 0 0 0 0 0
0 0 0 1 0 0 1 0      0 1 0 0 0 1 0 1      0 0 0 0 0 0 1 0      0 0 0 0 0 0 0 0
0 0 0 0 1 0 1 0      0 1 0 1 0 1 0 0      1 0 0 0 0 0 1 0      0 0 0 0 0 0 0 0
0 0 0 0 1 0 1 0      0 0 0 0 0 0 1 0      0 0 0 0 0 0 0 1      0 0 0 0 0 0 0 0

20                   21                   22                   23
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 1      0 0 0 1 0 1 0 1      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 1      0 0 0 0 0 0 0 1      0 0 0 0 0 0 0 0
1 1 1 1 1 0 0 0      0 0 0 0 0 0 0 0      0 0 0 1 0 1 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 0      0 0 0 0 0 0 0 0      0 0 0 1 0 0 0 1      0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0      0 0 0 0 0 0 1 0      1 0 0 1 0 0 0 1      0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0      0 0 0 0 0 0 1 0      1 0 0 1 0 0 0 1      0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0      0 0 0 0 0 0 0 0      1 0 0 1 0 1 0 1      0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0      0 0 0 0 0 1 0 0      1 0 0 1 0 1 0 1      0 0 0 0 0 0 0 0

24                   25                   26                   27
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0

28                   29                   30                   31
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 1 0 0 0 1      1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 1 0 0 0 1 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 1      1 1 0 0 1 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 1 1 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 1 0 0      0 1 1 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0

32                   33                   34                   35
0 0 0 0 1 0 1 0      0 0 0 0 1 0 1 0      0 0 0 0 0 0 0 1      1 1 0 0 0 0 0 0
0 0 0 0 1 0 1 0      1 0 0 0 1 0 1 1      0 1 0 0 0 0 0 0      0 0 1 0 1 1 1 1
0 0 0 0 1 0 1 0      1 0 0 0 1 0 0 1      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 1 0 1 0      1 0 0 1 0 0 0 0      1 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 1 0 1 0      1 0 0 1 0 1 0 0      1 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 1 0 0 1 0      1 0 0 1 0 1 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 1 0 1 0      0 0 0 0 0 1 0 0      0 1 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 1 0 0 0 1 0      0 1 1 0 0 0 0 0      0 1 1 0 0 0 0 0      0 0 0 0 1 1 0 1

36                   37                   38                   39
0 0 0 0 0 1 1 0      0 0 0 0 0 1 0 1      0 0 0 1 0 0 0 1      0 0 0 0 0 0 0 0
1 1 1 1 1 1 0 0      0 0 0 0 0 1 0 1      0 0 0 1 0 1 0 1      0 0 0 0 0 0 0 0
0 0 0 1 1 0 0 0      0 0 0 0 0 0 0 1      0 1 0 1 0 1 0 1      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 1      0 1 0 1 0 1 0 1      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 1      0 1 0 1 0 1 0 1      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 1      0 0 0 1 0 1 0 1      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 1      0 0 0 0 0 0 0 1      0 0 0 0 0 0 0 0
```

```
1 1 1 1 1 1 0 0      0 0 0 0 0 0 0 1      0 0 0 1 0 1 0 1      0 0 0 0 0 0 0 0

40                   41                   42                   43
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0

44                   45                   46                   47
0 0 0 0 0 0 0 0      0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 1 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      1 1 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 1 1 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0

48                   49                   50                   51
0 0 1 1 0 0 1 1      0 1 1 1 1 1 0 0      0 0 1 1 0 0 0 0      0 0 1 1 1 0 0 0
0 0 0 1 0 0 1 0      0 0 0 0 0 1 1 0      0 0 0 0 0 0 0 0      0 0 1 0 0 0 0 0
0 0 0 1 0 0 1 1      0 0 0 0 0 0 0 0      0 0 1 0 1 0 1 1      1 0 0 0 0 0 0 0
0 0 0 1 0 0 1 0      0 0 0 1 1 0 0 0      0 0 0 0 0 0 0 0      0 0 0 1 0 0 0 0
0 0 0 1 0 0 1 0      0 1 1 0 1 0 1 1      0 0 1 0 0 1 0 0      0 0 1 1 0 0 0 0
0 0 0 1 0 0 1 0      1 0 1 0 0 0 0 0      1 1 1 1 0 1 0 1      1 0 1 0 1 1 1 1
0 0 0 1 0 0 1 0      1 0 1 0 0 1 0 0      0 0 0 0 0 1 0 0      0 0 0 0 0 0 0 0
0 0 0 1 0 1 0 0      1 0 0 1 0 0 0 0      0 0 0 1 0 1 0 0      0 0 0 1 0 1 0 0

52                   53                   54                   55
0 0 0 0 0 0 1 0      0 0 0 0 0 0 0 1      0 0 0 1 0 0 0 1      0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1      0 0 0 0 0 0 0 1      0 0 1 1 1 1 0 1      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1      0 0 0 1 1 0 1 1      0 0 1 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 1 0      0 0 0 0 1 0 1 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0      0 0 1 0 0 0 0 0      0 0 0 1 0 0 1 0      0 0 0 0 0 0 0 0
1 0 0 1 1 1 1 1      1 1 1 1 1 0 0 1      0 0 1 0 0 0 1 0      0 0 0 0 0 0 0 0
1 1 0 0 0 0 0 0      0 0 0 0 1 1 0 1      0 0 1 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 1      0 0 0 0 0 0 1 0      0 0 0 0 0 0 0 0

56                   57                   58                   59
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 1      1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 1 0      0 0 0 0 0 0 0 0

60                   61                   62                   63
```

```
1 1 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0

64                 65                 66                 67
0 0 0 1 0 1 0 0    1 0 0 1 0 0 0 0    0 0 0 1 0 1 0 0    0 0 0 0 0 0 0 0
0 0 0 1 0 1 0 0    1 0 1 0 0 1 0 0    0 0 0 1 0 1 0 0    0 0 0 0 0 0 0 0
0 0 0 1 0 1 0 0    1 0 0 0 0 1 0 0    0 0 0 1 0 1 0 0    0 0 0 0 0 0 0 0
0 0 0 1 0 1 0 0    0 0 1 0 0 1 0 1    0 0 0 1 0 1 0 0    0 0 0 0 0 0 0 0
0 0 0 1 0 1 0 0    1 0 1 0 1 0 0 1    1 0 0 1 0 1 0 0    0 0 0 0 0 0 0 0
0 0 0 1 0 1 0 0    0 0 1 0 1 0 0 0    1 0 0 1 0 1 0 0    0 0 0 0 0 0 0 0
0 0 0 1 0 1 0 0    0 0 0 0 1 0 0 0    1 0 0 0 0 1 0 0    0 0 0 0 0 0 0 0
0 0 0 1 0 1 0 0    0 0 0 0 1 0 0 0    1 0 1 0 0 1 0 0    0 0 0 0 0 0 0 0

68                 69                 70                 71
0 0 0 0 0 1 0 0    0 1 0 0 0 1 0 0    0 0 1 0 0 0 0 1    0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0    0 0 0 0 0 1 0 0    0 0 1 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0    1 1 1 0 0 1 0 0    0 0 0 1 0 0 0 1    0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0    0 0 1 0 0 1 0 1    0 0 1 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0    1 0 0 0 0 1 0 0    0 0 1 0 1 0 1 0    0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0    1 0 0 0 0 1 0 1    0 0 1 0 1 0 1 0    0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0    0 0 0 0 1 1 0 1    0 0 1 0 1 0 1 0    0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0    0 0 0 0 1 0 0 1    0 1 0 0 0 0 1 0    0 0 0 0 0 0 0 0

72                 73                 74                 75
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0

76                 77                 78                 79
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0    0 0 0 0 0 0 0 0

80                 81                 82                 83
0 0 0 1 0 0 0 0    0 0 0 0 1 0 0 0    1 0 1 0 0 1 0 0    0 0 0 0 0 0 0 0
0 0 0 1 0 1 0 0    0 0 0 0 1 0 0 0    1 0 1 0 1 0 0 0    0 0 0 0 0 0 0 0
0 0 0 1 0 1 0 0    0 0 0 0 1 0 0 0    1 0 1 0 1 0 0 0    0 0 0 0 0 0 0 0
```

```
0 0 0 1 0 0 0 0      0 0 0 0 1 0 0 0      1 1 0 0 1 0 0 0      0 0 0 0 0 0 0 0
0 0 1 1 0 1 0 0      0 0 0 0 0 0 0 1      0 1 0 0 1 0 0 0      0 0 0 0 0 0 0 0
0 0 0 1 0 0 1 0      0 0 0 0 1 0 0 1      0 1 0 0 1 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0      0 0 0 0 1 0 0 1      0 1 0 0 1 0 0 0      0 0 0 0 0 0 0 0
0 0 0 1 0 1 0 0      0 0 0 0 1 0 0 1      0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0

84                   85                   86                   87
0 0 0 0 0 0 0 0      0 0 0 0 1 0 0 1      0 0 0 0 1 0 1 0      0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0      0 0 0 0 1 0 0 1      0 1 0 1 0 0 1 0      0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 1      0 0 0 0 1 0 0 1      0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 1 1 0 1      0 0 0 0 1 0 0 1      0 0 0 1 0 0 1 0      0 0 0 0 0 0 0 0
0 0 0 1 1 0 0 1      0 0 0 0 1 0 0 1      0 0 0 0 1 0 1 0      0 0 0 0 0 0 0 0
0 0 0 1 1 0 0 1      0 0 0 0 1 0 0 0      0 0 0 1 0 0 1 0      0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 1      0 0 0 0 0 1 0 1      0 0 0 1 0 0 1 0      0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 1      0 0 0 0 0 1 0 0      0 0 0 1 0 0 1 0      0 0 0 0 0 0 0 0

88                   89                   90                   91
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0

92                   93                   94                   95
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0

96                   97                   98                   99
0 0 0 0 0 1 0 0      0 0 0 0 1 0 0 1      0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0
0 0 1 0 0 1 1 0      0 0 0 0 1 0 1 0      0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0      0 0 0 0 1 0 1 0      0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0
0 0 1 0 0 1 0 0      0 0 0 0 0 0 1 0      0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0
0 0 1 0 0 1 0 0      0 0 0 0 0 0 0 0      0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0
0 0 1 0 0 1 1 0      0 0 0 0 0 0 0 0      0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0
0 0 1 0 0 1 1 0      0 0 0 0 0 0 0 0      0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0

100                  101                  102                  103
0 0 0 0 0 1 0 0      0 0 0 0 0 1 0 0      0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 1      0 0 0 0 0 1 0 0      0 0 0 1 0 0 1 0      0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0      0 0 0 0 0 0 0 0      0 0 0 1 0 0 1 0      0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0      0 0 0 0 0 0 1 0      0 0 0 1 0 0 1 0      0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0      0 0 0 0 0 0 0 0      0 0 0 1 0 0 1 0      0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0      0 0 0 0 1 0 0 0      0 0 0 1 0 0 1 0      0 0 0 0 0 0 0 0
```

```
0 0 0 0 1 0 0 0     0 0 0 0 0 0 0 0     0 0 1 0 0 0 1 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 1 0 0 0     0 0 1 0 0 0 0 0     0 0 0 0 0 0 0 0

104                 105                 106                 107
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0


108                 109                 110                 111
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0

112                 113                 114                 115
0 0 1 0 0 1 0 0     0 0 0 0 0 0 0 0     0 0 0 1 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0     0 0 0 0 0 0 0 0     0 0 0 1 0 0 0 0     0 0 0 0 0 0 0 0
0 0 1 0 0 1 0 0     0 0 0 0 0 0 0 0     0 0 0 1 0 0 0 0     0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 0     0 0 0 0 0 0 0 0     0 0 0 1 0 0 0 0     0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 0     0 0 0 0 0 0 0 0     0 0 0 1 0 0 0 0     0 0 0 0 0 0 0 0
0 1 0 0 1 0 0 0     0 0 0 0 0 0 0 0     0 0 0 1 0 0 0 0     0 0 0 0 0 0 0 0
0 1 0 1 1 1 0 0     0 0 0 0 0 0 0 0     0 0 0 1 0 0 0 0     0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0     0 0 0 0 0 0 0 1     1 1 1 1 0 0 0 0     0 0 0 0 0 0 0 0

116                 117                 118                 119
0 0 0 0 1 0 0 0     0 0 0 0 1 0 0 0     0 0 0 1 0 0 1 0     0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0     0 0 0 0 1 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 1 0 1 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 1 1 0 0     0 0 0 0 1 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     1 1 1 0 1 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1     1 1 1 0 0 1 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0     0 0 1 1 0 1 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0     0 0 0 0 0 1 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0

120                 121                 122                 123
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0     0 0 0 0 1 1 1 0     0 0 0 0 0 0 0 0     0 0 0 0 0 1 0 1
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
```

```
124                  125                  126                  127
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
1 1 0 0 0 0 1 1      1 0 0 0 0 0 0 1      1 1 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 1 0 0 0 0 0      0 0 0 0 0 0 0 0

128                  129                  130                  131
0 0 0 0 0 0 0 0      0 0 0 1 1 1 0 1      0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1      1 1 0 0 0 0 0 0      0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0
0 1 1 0 0 0 0 0      0 0 0 1 0 1 1 1      0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0      0 0 0 0 0 0 0 0      0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 1 1 0 0 0      0 0 0 0 0 0 0 0      0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 1 1 0 0 0      0 0 0 0 0 0 0 0      0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0

132                  133                  134                  135
0 0 0 0 1 0 1 0      0 0 0 1 0 1 1 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0      0 0 0 1 0 0 0 0      1 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0      0 0 0 0 0 1 0 0      1 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0      0 0 1 0 0 1 0 0      0 0 1 1 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0      1 1 1 0 1 0 0 1      0 0 0 0 1 1 1 0      0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0      0 0 0 0 1 0 0 0      1 1 1 0 0 0 0 1      0 0 0 0 0 1 0 0
0 0 0 0 1 0 0 0      0 0 0 0 1 0 0 0      0 0 1 0 0 0 0 1      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 1 0 0 0      0 0 0 0 1 0 0 0      1 0 0 0 0 0 0 0

136                  137                  138                  139
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0

140                  141                  142                  143
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0

144                  145                  146                  147
0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0      0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0
```

```
0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0      0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0      0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0      0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0      0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0      0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0

148                  149                  150                  151
0 0 0 0 0 0 0 0      0 0 0 0 1 0 0 0      0 0 0 0 0 1 0 0      0 0 1 1 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 1      0 0 0 0 1 1 0 0
0 0 0 0 0 0 0 0      0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0      1 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0      0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0      0 1 0 0 0 0 0 1
0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 1 0 0 0 0
0 0 0 0 1 0 0 0      0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 1 0 0
0 0 0 1 0 1 1 0      0 0 0 0 1 1 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 1

152                  153                  154                  155
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
1 1 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 1 1 1 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
1 0 0 0 0 1 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 1 1 0 0 0      1 1 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0

156                  157                  158                  159
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0

160                  161                  162                  163
0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0      0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0      0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0      0 0 0 0 0 0 1 0      0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0      0 0 0 1 0 0 1 0      0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0      0 0 0 1 0 0 1 0      0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0      0 0 0 0 0 0 1 0      0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0      0 0 0 1 0 0 1 0      0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0      0 0 0 1 0 0 1 0      0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0

164                  165                  166                  167
0 0 0 1 0 0 1 0      0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0      0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0      0 0 0 1 1 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0      0 0 0 1 0 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0      0 0 0 0 1 0 0 0      0 0 0 0 0 0 0 0      0 0 0 0 0 0 0 0
```

```
0 0 0 1 0 0 0 0     0 0 0 0 1 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0     0 0 0 0 1 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0

168                 169                 170                 171
1 0 0 0 0 0 0 0     0 0 1 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 1 0 0 0 0 1 1     0 0 0 1 1 1 1 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0     0 0 0 0 0 0 1 1     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0     0 1 0 0 0 0 0 1     0 1 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1     0 0 0 1 1 0 0 0     0 1 0 1 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0     1 0 0 0 0 1 1 0     0 0 0 0 1 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     1 0 1 0 0 0 0 0     0 0 0 0 0 1 1 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 1 0 1 0 0 0     1 0 0 0 0 0 0 1     1 0 0 0 0 0 0 0

172                 173                 174                 175
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0

176                 177                 178                 179
0 0 0 0 1 0 0 0     0 0 0 1 0 0 1 0     0 0 0 0 1 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 1 0 0 1 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 1 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0     0 0 0 1 0 0 0 0     0 0 0 0 1 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0     0 0 0 1 0 0 0 0     0 0 0 0 1 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0     0 0 0 1 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 1
0 0 0 0 1 1 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 1 1 1 1 0 1 1
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0

180                 181                 182                 183
0 0 0 1 1 0 0 0     0 0 0 1 1 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0     0 0 0 0 1 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 1 1 1 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0     0 0 0 0 0 0 1 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0     0 0 0 0 1 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 1 1 1 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0

184                 185                 186                 187
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 1 1 0 0 0 0     0 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 1 1 1     0 0 0 0 1 1 0 0     0 0 0 1 1 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     1 0 0 0 0 0 0 0     0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 1 1 1 0 0 0 1     1 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 1 0 0 0     0 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 1 0 0 0     0 0 1 1 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0
```

188	189	190	191
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 1 1 1 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1	1 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0

Results Vectors

The following table shows result vectors for sub-images shown above for a test image. In each column there is a sub-column for sub-image ID followed by the eight binary numbers indicating whether or not that there is a vertical, vertical backslash, backslash, horizontal backslash, horizontal, horizontal slash, slash, and vertical slash respectively.

Sub-Image Number /Result Vector		**Sub-Image Number /Result Vector**		**Sub-Image Number /Result Vector**	
0	1 1 0 0 0 0 0 1	64	1 1 1 0 0 0 0 1	128	1 1 0 1 1 0 1 1
1	0 1 0 1 1 1 0 0	65	1 1 1 1 1 1 1 1	129	0 0 0 0 0 0 0 0
2	0 0 0 0 0 0 0 0	66	1 1 0 0 0 0 0 1	130	1 1 1 0 0 0 0 1
3	0 0 0 0 0 0 0 0	67	0 0 0 0 0 0 0 0	131	0 0 0 0 0 0 0 0
4	0 0 0 0 0 0 0 0	68	1 1 0 0 0 0 0 1	132	1 1 1 0 0 0 0 1
5	0 0 0 0 0 0 0 0	69	1 0 0 0 1 1 0 1	133	1 0 0 1 1 1 1 0
6	1 1 0 1 0 1 0 1	70	1 1 1 0 0 1 0 1	134	1 1 1 1 1 0 0 1
7	0 0 1 1 0 1 1 1	71	0 0 0 0 0 0 0 0	135	0 0 0 0 0 0 0 0
8	0 0 0 0 0 0 0 0	72	0 0 0 0 0 0 0 0	136	0 0 0 0 0 0 0 0
9	0 0 0 0 0 0 0 0	73	0 0 0 0 0 0 0 0	137	0 0 0 0 0 0 0 0
10	0 0 0 0 0 0 0 0	74	0 0 0 0 0 0 0 0	138	0 0 0 0 0 0 0 0
11	0 0 0 0 0 0 0 0	75	0 0 0 0 0 0 0 0	139	0 0 0 0 0 0 0 0
12	0 0 0 0 0 0 0 0	76	0 0 0 0 0 0 0 0	140	0 0 0 0 0 0 0 0
13	0 0 0 0 0 0 0 0	77	0 0 0 0 0 0 0 0	141	0 0 0 0 0 0 0 0
14	0 0 0 0 0 0 0 0	78	0 0 0 0 0 0 0 0	142	0 0 0 0 0 0 0 0
15	0 0 1 1 1 1 1 1	79	0 0 0 0 0 0 0 0	143	0 0 0 0 0 0 0 0
16	1 1 0 1 1 1 0 1	80	1 1 1 0 0 0 0 1	144	1 1 0 0 0 0 0 0
17	1 1 0 0 1 1 0 1	81	1 1 0 0 0 0 0 0	145	0 0 0 0 0 0 0 0
18	0 1 0 0 0 1 0 0	82	1 1 0 0 0 1 0 1	146	1 1 1 0 0 0 0 1
19	0 0 0 0 1 0 0 0	83	0 0 0 0 0 0 0 0	147	0 0 0 0 0 0 0 0
20	0 0 0 1 1 1 0 0	84	1 1 1 1 1 1 1 1	148	0 0 1 0 0 0 0 0
21	0 0 0 0 0 0 0 1	85	1 1 1 0 0 0 0 0	149	1 0 0 0 0 0 0 0
22	1 0 0 0 0 0 0 1	86	1 1 1 0 0 1 0 1	150	0 0 0 0 0 0 0 0
23	0 0 0 0 0 0 0 0	87	0 0 0 0 0 0 0 0	151	1 1 1 1 1 1 1 0
24	0 0 0 0 0 0 0 0	88	0 0 0 0 0 0 0 0	152	0 0 0 1 1 1 1 1
25	0 0 0 0 0 0 0 0	89	0 0 0 0 0 0 0 0	153	0 0 0 0 0 0 0 0
26	0 0 0 0 0 0 0 0	90	0 0 0 0 0 0 0 0	154	0 0 0 0 0 0 0 0
27	0 0 0 0 0 0 0 0	91	0 0 0 0 0 0 0 0	155	0 0 0 0 0 0 0 0
28	0 0 0 0 0 0 0 0	92	0 0 0 0 0 0 0 0	156	0 0 0 0 0 0 0 0

29	0 0 0 0 0 0 0 0	93	0 0 0 0 0 0 0 0	157	0 0 0 0 0 0 0 0
30	1 1 0 1 1 0 1 1	94	0 0 0 0 0 0 0 0	158	0 0 0 0 0 0 0 0
31	0 0 0 1 0 1 0 0	95	0 0 0 0 0 0 0 0	159	0 0 0 0 0 0 0 0
32	1 1 0 0 1 1 0 1	96	1 1 0 0 0 1 0 1	160	1 1 0 0 0 0 0 0
33	1 0 0 1 1 1 1 1	97	0 0 0 0 0 0 0 0	161	0 1 0 0 0 1 0 1
34	1 0 0 0 0 0 0 1	98	1 0 1 0 0 1 1 0	162	1 1 1 1 1 1 1 1
35	0 0 0 1 0 1 1 0	99	0 0 0 0 0 0 0 0	163	0 0 0 0 0 0 0 0
36	1 0 0 1 1 1 1 1	100	1 1 0 0 0 0 0 1	164	1 1 0 0 0 0 0 0
37	1 1 0 0 0 0 0 0	101	0 0 0 0 1 0 0 0	165	1 1 1 0 0 0 1 0
38	1 1 1 0 0 0 0 1	102	1 1 0 0 0 0 0 1	166	0 0 0 0 0 0 0 0
39	0 0 0 0 0 0 0 0	103	0 0 0 0 0 0 0 0	167	0 0 0 0 0 0 0 0
40	0 0 0 0 0 0 0 0	104	0 0 0 0 0 0 0 0	168	0 0 0 0 0 0 0 0
41	0 0 0 0 0 0 0 0	105	0 0 0 0 0 0 0 0	169	1 1 0 1 1 1 1 0
42	0 0 0 0 0 0 0 0	106	0 0 0 0 0 0 0 0	170	1 0 0 1 1 0 0 0
43	0 0 0 0 0 0 0 0	107	0 0 0 0 0 0 0 0	171	0 0 0 0 0 0 0 0
44	0 0 0 0 0 0 0 0	108	0 0 0 0 0 0 0 0	172	0 0 0 0 0 0 0 0
45	0 1 0 0 0 0 1 0	109	0 0 0 0 0 0 0 0	173	0 0 0 0 0 0 0 0
46	0 0 0 0 0 0 0 0	110	0 0 0 0 0 0 0 0	174	0 0 0 0 0 0 0 0
47	0 0 0 0 0 0 0 0	111	0 0 0 0 0 0 0 0	175	0 0 0 0 0 0 0 0
48	1 1 0 0 0 1 0 1	112	1 1 1 0 1 1 1 1	176	1 1 1 0 0 0 0 1
49	1 1 0 1 1 1 0 1	113	0 0 0 0 0 0 0 0	177	0 0 0 0 0 0 0 1
50	1 1 0 0 1 1 1 1	114	1 1 0 0 0 0 0 1	178	1 1 0 0 0 0 0 0
51	1 1 0 1 1 0 0 1	115	0 0 0 0 0 0 0 0	179	0 0 0 0 1 1 0 0
52	0 0 0 0 1 1 1 1	116	1 0 1 1 1 1 0 0	180	0 0 1 0 0 1 1 0
53	1 0 1 0 0 1 1 0	117	1 1 0 1 1 0 1 1	181	1 0 0 0 0 0 0 0
54	0 1 0 1 1 1 1 1	118	0 0 0 0 1 0 0 0	182	0 0 0 0 0 0 0 0
55	0 0 0 0 0 0 0 0	119	0 0 0 0 0 0 0 0	183	0 0 0 0 0 0 0 0
56	0 0 0 0 0 0 0 0	120	0 0 0 0 0 0 0 0	184	0 0 0 0 0 0 0 0
57	0 0 0 0 0 0 0 0	121	0 0 0 0 1 0 0 0	185	0 0 0 0 1 0 0 0
58	0 0 0 0 0 0 0 0	122	0 0 0 0 0 0 0 0	186	1 0 1 1 1 0 0 0
59	0 0 1 0 0 0 0 0	123	0 0 0 0 1 0 0 0	187	1 1 1 1 1 1 0 0
60	0 0 1 0 0 0 0 0	124	0 0 0 0 0 1 0 0	188	0 0 1 0 1 1 0 0
61	0 0 0 0 0 0 0 0	125	0 0 1 0 0 0 0 0	189	0 0 0 0 0 0 0 0
62	0 0 0 0 0 0 0 0	126	0 0 0 0 0 0 1 0	190	0 0 0 0 0 0 0 0
63	0 0 0 0 0 0 0 0	127	0 0 0 0 0 0 0 0	191	0 0 0 0 0 0 0 0

Appendix E Sample Source Code

File “point.h” Contents

```
/*point Class Definition*/
class point
{
private:
    int id;
    int intensity, edgeBinaryValue;
    float gradientMagnitude, gradientAngle;

public:
    void setID(int ID){id = ID;}
    void setIntensity(int INTENSITY){intensity = INTENSITY;}
    int getID(){return id;}
    int getIntensity(){return intensity;}
    void setGradientMagnitude(float GM){gradientMagnitude = GM;}
    float getGradientMagnitude(){return gradientMagnitude;}
    void setGradientAngle(float ANGLE){gradientAngle = ANGLE;}
    float getGradientAnlge(){return gradientAngle;}
    void setEdgeBinaryValue(int EBV){edgeBinaryValue = EBV;}
    int getEdgeBinaryValue(){return edgeBinaryValue;}
};
```

File “HT_point.h” Contents

```
#include <fstream.h>
#include <math.h>
#include "line.h"

class HT_point
{
private:
    int id;
    int theta;
    int rho;
    int accumulation;
    int clusterID;
    float distance;
    float m,c;
    int category;
    int isPeak, isBFPeak_3x3, isBFPeak_5x5;     //mark acc array points
as peak after thresholding
    int *contributingPoints;
    int indexToContPoints;
    int *lineIn, *h14lineIn;
    line *ln; //8 being the maximum possible number of lines
    int countOfLongLines;

public:
    HT_point();
    HT_point(int ID);
    HT_point(int THETA,int RHO);
    HT_point(int ID,int THETA,int RHO);

    void setTheta(int THETA){theta = THETA;}
    void setRho(int RHO){rho = RHO;}
    void setID(int ID){id = ID;}
    void setClusterID(int CLUSTER_ID){clusterID = CLUSTER_ID;}
    void setDistance(double DISTANCE){distance = DISTANCE;}
    int getTheta(void){return theta;}
    int getRho(void){return rho;}
    int getID(void){return id;}
    float get_c(){return c;}
    float get_m(){return m;}
    int getClusterID(void){return clusterID;}
    void increaseAccumulation(void){accumulation++;}
    void setAccumulation(int ACCUMULATION){accumulation=ACCUMULATION;}
    int getAccumulation(void){return accumulation;}


    void setIsPeak(int P){isPeak = P;}
    bool getIsPeak(){return isPeak;}
    void setIsBFPeak_3x3(int P){isBFPeak_3x3 = P;}
    bool getIsBFPeak_3x3(){return isBFPeak_3x3;}
    void setIsBFPeak_5x5(int P){isBFPeak_5x5 = P;}
    bool getIsBFPeak_5x5(){return isBFPeak_5x5;}

    void setUpContPointsArray();
    void addContPoint(int IMAGE_POINT_ID);
    void printContPoints(ofstream& OFS);
    void h14printContPoints(ofstream& OFS);

    int formLines();
    void printLines(ofstream& OFS);
    int getNumOfSubLines(){return countOfLongLines;};

    int h14formLines();
};
```

File "image.h" contents

```
/*image Class Definition*/
#include "point.h"
#include "HT_point.h"
#include <fstream.h>
#include "subResults.h"

class image
{
private:
    int id, edgeThreshold;
    float GMMax, GMMin;          //gradient magnitude max and min (used
for auto threshold determination)
    point *p;                    //image points
    HT_point *psp;               //parameter space points
    HT_point *bfp;               //butterfly filtered points
    int *maxima, *maxIndices, *maxima_5x5, *maxIndices_5x5;   //to hold
info about peaks found
    int maxCount, maxCount_5x5;

    int *groupCount;             //array to hold info about groups during
statistical threshold determination

    int peakCount;

    int numOfLinesWithSubLines, numOfSubLines;

    bool itIs5x5Peak;

    HT_point *h14psp;
    point *h14_p;
    results subRlts;
    int h14peakCount;
    HT_point *h14bfp;

public:
    image();
    void printID();
    void setIntensity(int ID, int
INTENSITY){p[ID].setIntensity(INTENSITY);}
    void printIntensity();

    void determineGradientMagnitude();
    void printGradientMagnitude();
    void determineEdgeDetectionThreshold();
    void printGradientAngle();
    int getEdgeThreshold(){return edgeThreshold;}
    int applyEdgeDetectionThreshold();
    void printEdgeBinaryValue();
    void printThinnedImage();

    void NewEdgeThining();
    int determineNumOfEdgesInNeighbourhood(int INDEX);
    void rankNeighbourhood(int INDEX, int neighbourhoodRankArray[]);
    bool checkStep2Condition(int INDEX, int neighbourhoodRankArray[]);
    void step2_1(int INDEX);
    bool checkStep2Condition2(int INDEX, int neighbourhoodRankArray[]);
    void step2_2(int INDEX);
    bool checkStep2Condition3(int INDEX, int neighbourhoodRankArray[]);
    void step2_3(int INDEX, int neighbourhoodRankArray[]);
    void step2_2_1(int INDEX);
    void step2_3_1(int INDEX);
    void step2_3_2(int INDEX);
    int determineClosestAngle(int INDEX);

    void HoughTransform();
    void updateAccArray(int THETA, float ROH);
    void printAccArray();

    int determineAAThreshold(int T);
    void peakDetect(int T);
    int determineTargetNumOfLines();
    void groupAAEntries(void);
    int determineNumberOfAAGroups();
    int findMaxAAEntry(void);
    int findMinAAEntry(void);
    void printPeaks();
    int getPeakCount(){return peakCount;}

    void updateContPointsArray(int theta, float roh, int i);
    void buildContributingPointsArray();

    void butterflyFilter();
    void printBFAccArray();
    int findBFLocalMaxima();
    int findBFLocalMaxima_5x5();
    int isBFPeak(int INDEX);
    int isBFPeak_5x5(int INDEX);
    void printBFPeaks();
    void printBFPeaks_5x5();
    void printBinaryBFAccArray_5x5();

    void printBFPeaks_5x5WithContPoints();

    void formSubLines();
    void printBFPeaks_5x5WithSubLines();
    void printBFPeaks_5x5WithSubLines_ForThesis();
    int getNumOfSubLines(){ return numOfSubLines;}
    int getNumOfLinesWithSubLines(){return numOfLinesWithSubLines;}

    //int findVanishingPoint();

    void Hybrid14HT(results sRES, int CAT_CODE);
    void updateh14AccArray(int THETA, float ROH);
    void h14printAccArray();

    void h14updateContPointsArray(int THETA, float ROH,int I);
    void h14buildContributingPointsArray(results SUB_RESULTS, int
CAT_CODE);


    void h14peakDetect(int T);
    int h14determineAAThreshold(int T);
    int h14findMaxAAEntry();
    int h14findMinAAEntry();
    void h14groupAAEntries(void);
    int h14getPeakCount(){return h14peakCount;}
    void h14printPeaks();

    void h14butterflyFilter();
    void h14printBFAccArray();
    int h14findBFLocalMaxima();
    int h14findBFLocalMaxima_5x5();
    int h14isBFPeak(int INDEX);
    int h14isBFPeak_5x5(int INDEX);
    void h14printBFPeaks();
    void h14printBFPeaks_5x5();
    void h14printBinaryBFAccArray_5x5();

    void h14printBFPeaksWithContPoints();
    void h14printBFPeaks_5x5WithContPoints();

    void h14formSubLines();
    void h14printBFPeaksWithSubLines();
};
```

File “subResults.h” Contents

```
class subResults
{
private:
    int id;
    int results[8]; //0=v,1=vBS,2=BS,3=hBS,4=h,5=hS,6=S,7=vS
public:
    void setID(int ID){id =ID;}
    void setVer(int V){results[0] = V;}
    void setVerBS(int V){results[1] = V;}
    void setBSlash(int V){results[2] = V;}
    void setHorBS(int V){results[3] = V;}
    void setHor(int V){results[4] = V;}
    void setHorSlash(int V){results[5] = V;}
    void setSlash(int V){results[6] = V;}
    void setVerS(int V){results[7] = V;}

    int getID(){return id;}
    int getVer(){return results[0];}
    int getVerBS(){return results[1];}
    int getBSlash(){return results[2];}
    int getHorBS(){return results[3];}
    int getHor(){return results[4];}
    int getHorSlash(){return results[5];}
    int getSlash(){return results[6];}
    int getVerS(){return results[7];}

    void printResults(ofstream&);
    void loadResults(ifstream&);
};

void subResults::printResults(ofstream& OFS)
{
    for (int i = 0; i<8; i++)
        OFS << results[i] << " ";
    OFS << endl;
};

void subResults::loadResults(ifstream& IFS)
{
    //ofstream writeRes("subImageResults.txt");
    for (int i = 0; i<8; i++)
        IFS >> results[i];
};

class results
{
private:
    int id;
    subResults sRlts[192];
public:
    void setID(int ID){id =ID;}
    void setVer(int V, int sID){sRlts[sID].setVer(V);}
    void setVerBS(int V, int sID){sRlts[sID].setVerBS(V);}
    void setBSlash(int V, int sID){sRlts[sID].setBSlash(V);}
    void setHorBS(int V, int sID){sRlts[sID].setHorBS(V);}
    void setHor(int V, int sID){sRlts[sID].setHor(V);}
    void setHorSlash(int V, int sID){sRlts[sID].setHorSlash(V);}
    void setSlash(int V, int sID){sRlts[sID].setSlash(V);}
    void setVerS(int V, int sID){sRlts[sID].setVerS(V);}

    int getID(){return id;}
    int getVer(int sID){return sRlts[sID].getVer();}
    int getVerBS(int sID){return sRlts[sID].getVerBS();}
    int getBSlash(int sID){return sRlts[sID].getBSlash();}
    int getHorBS(int sID){return sRlts[sID].getHorBS();}
    int getHor(int sID){return sRlts[sID].getHor();}
    int getHorSlash(int sID){return sRlts[sID].getHorSlash();}
    int getSlash(int sID){return sRlts[sID].getSlash();}
    int getVerS(int sID){return sRlts[sID].getVerS();}

    void printResults(ofstream&);
    void loadResults(ifstream&);
};

void results::printResults(ofstream& OFS)
{
    OFS << id << endl;
    for (int i=0; i<192; i++)
    {
        //OFS << i << "\t";
        sRlts[i].printResults(OFS);
    }
};

void results::loadResults(ifstream& IFS)
{
    IFS >> id;
    for (int i=0; i<192; i++)
    {
        sRlts[i].loadResults(IFS);
    }
};
```

Some Main Source File Contents

```
//---------------------------------------------------------------------------
#include <time.h> // for timing functions
#include <fstream.h> //for file input and output (mostly for debugging)
#include "image.h"
#include "pattern.h"
#include"backprop.h"
#include"common.h"
//---------------------------------------------------------------------------
TMainForm *MainForm;

image I;
results sResults[1];

const int glb_X_LOWER_BOUND_ABS_VAL = 64;
const int glb_X_UPPER_BOUND = 63;
const int glb_Y_LOWER_BOUND_ABS_VAL = 48;
const int glb_Y_UPPER_BOUND = 47;

const float glb_CONVERT = 3.14159265/180;
//---------------------------------------------------------------------------
void __fastcall TMainForm::CaptureSingleImage1Click(TObject *Sender)
{
    VLSnapshot1->Snapshot();
    Image1->Refresh();
}

//---------------------------------------------------------------------------
void __fastcall TMainForm::SobelEdgeDetection1Click(TObject *Sender)
{
/*****************************************************************
*
*   This function does edge detection image.
*
*****************************************************************/
    clock_t start, end;
    start = clock();

    I.determineGradientMagnitude();
    I.determineEdgeDetectionThreshold();
    int numOfEdgePoints = I.applyEdgeDetectionThreshold();

    end = clock();
    float timeTaken = (end - start)/CLK_TCK;
    Memo1->Lines->Add("Edge-detection completed in " +
FloatToStr(timeTaken)+"secs");
    Memo1->Lines->Add("   - threshold is " +
IntToStr(I.getEdgeThreshold())
                            + " and number of edge points if " +
numOfEdgePoints);

    I.printGradientMagnitude();
    I.printGradientAngle();
    I.printEdgeBinaryValue();
}
```

```
//---------------------------------------------------------------------------

void __fastcall TMainForm::ModifiedEdgeThining1Click(TObject *Sender)
{
    clock_t start, end;
    start = clock();

    I.NewEdgeThining();

    end = clock();
    float timeTaken = (end - start)/CLK_TCK;

    Memo1->Lines->Add("Thinning completed in " +
FloatToStr(timeTaken)+"secs");

    I.printThinnedImage();
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::HoughTransformwithPolarCoordinates1Click(
      TObject *Sender)
{
    clock_t start, end;
    start = clock();

    I.HoughTransform();
    I.buildContributingPointsArray();

    end = clock();
    float timeTaken = (end - start)/CLK_TCK;

    Memo1->Lines->Add("Hough transform completed in " +
FloatToStr(timeTaken)+"secs");

    I.printAccArray();
}
//---------------------------------------------------------------------------
void __fastcall TMainForm::RunComplete1Click(TObject *Sender)
{
    TObject *s;
    LoadtoImage11Click(s);
    De1Click(s);        //Determine intensities
    SobelEdgeDetection1Click(s);
    SimulateEdgeImage1Click(s);
    ModifiedEdgeThining1Click(s);
    SimulateThinnedImage1Click(s);
    HoughTransformwithPolarCoordinates1Click(s);
    PeakDetect1Click(s);
    ApplyReducedButterflyFilter1Click(s);
    EndPointsDetermination1Click(s);
}
//---------------------------------------------------------------------------
void __fastcall TMainForm::ApplyReducedButterflyFilter1Click(
      TObject *Sender)
{
    ofstream writePeaks("BFPeaksInAA.txt");
    clock_t start, end;
    start = clock();

    I.butterflyFilter();

    int numOfBFPeaks = I.findBFLocalMaxima();
    int numOfBFPeaks_5x5 = I.findBFLocalMaxima_5x5();
    end = clock();
    float timeTaken = (end - start)/CLK_TCK;

    Memo1->Lines->Add("Reduced butterfly filter applied in " +
FloatToStr(timeTaken)+"secs");
    Memo1->Lines->Add("     - number of peaks (within 3x3
neighbourhoods) found is " + IntToStr(numOfBFPeaks));
    Memo1->Lines->Add("     - number of peaks (within 5x5
neighbourhoods) found is " + IntToStr(numOfBFPeaks_5x5));

    I.printBFAccArray();
    I.printBFPeaks();
    I.printBFPeaks_5x5();
    I.printBFPeaks_5x5WithContPoints();
    I.printBinaryBFAccArray_5x5();
    I.printPeaks();
}
//---------------------------------------------------------------------------

void __fastcall TMainForm::PeakDetect1Click(TObject *Sender)
{
    int target = StrToInt(InputBox("Target Number of Lines", "Please
supply target number of lines","200"));
    //int target = 200;

    clock_t start, end;
    start = clock();

    I.peakDetect(target);

    end = clock();
    float timeTaken = (end - start)/CLK_TCK;

    Memo1->Lines->Add("Peak detection completed in " +
FloatToStr(timeTaken)+"secs");
    Memo1->Lines->Add(" - number of peaks is " +
IntToStr(I.getPeakCount()));
    I.printPeaks();
}
//---------------------------------------------------------------------------
void __fastcall TMainForm::EndPointsDetermination1Click(TObject *Sender)
{
    clock_t start, end;
    start = clock();

    I.formSubLines();
    I.printBFPeaks_5x5WithSubLines();
    I.printBFPeaks_5x5WithSubLines_ForThesis();

    end = clock();
    float timeTaken = (end - start)/CLK_TCK;

    Memo1->Lines->Add("Sub lines determined in " +
FloatToStr(timeTaken)+"secs");
    Memo1->Lines->Add("   - number of lines with valid sublines is " +
IntToStr(I.getNumOfLinesWithSubLines()));
    Memo1->Lines->Add("   - number of sublines is " +
IntToStr(I.getNumOfSubLines()));
    ofstream writeNumOfSubLines("NumOfSubLines.txt");
    writeNumOfSubLines << I.getNumOfSubLines();
}
//---------------------------------------------------------------------------
void __fastcall TMainForm::Train1Click(TObject *Sender)
{
    clock_t start, end;
    start = clock();
    float timeTaken;

    srand(123);
    const int NUM_IN_TRAIN_SET = 56;
    const int DATA_WIDTH = 8;
    const int DATA_HEIGHT = 96;
    const int INPUT_SIZE = DATA_WIDTH*DATA_HEIGHT;   //stripe of width
8, height 96
    const int OUTPUT_SIZE =1;
    ifstream read_train("TrainingSet_8x96In_1Out.txt");
    ofstream ScreenToFile("TrainingFile_8x96In_1Out.txt",ios::app);

      // Create Training Set - XOR problem
    Pattern *train_data[NUM_IN_TRAIN_SET];
    //Pattern *test_data[NUM_IN_TEST_SET];

                                // sizes  id    input     output
                            // -----  --    -----     ------
    for (int i = 0; i < NUM_IN_TRAIN_SET; i++)
    {

     train_data[i]=new Pattern(INPUT_SIZE, OUTPUT_SIZE, read_train );
    }

     for (int i = 0; i< NUM_IN_TRAIN_SET; i++) train_data[i]-
>PrintToFile(ScreenToFile, DATA_WIDTH);

     // Create Backprop Network
     Backprop_Network a(0.45, 0.9,  4,   INPUT_SIZE,4*96,2*96,
OUTPUT_SIZE);   // 3 layers, INPUT_SIZE inputs, 3 middle, OUTPUT_SIZE
output
                                          // Learning rate 0.45, momemtum
0.9
// Train Backprop Network
     long iteration=0;
     int good=0;
     //int good2=0;  //to be based on total error rather than error on
each output
     double tolerance=0.5;
     //double tolerance2 = 0.5;
     double total_error;


    while (good<NUM_IN_TRAIN_SET)   // Train until all patterns are
correct
    {
        good=0;
        total_error=0.0;

        for (int i=0; i<NUM_IN_TRAIN_SET; i++)
            {
                a.Set_Value(train_data[i]);  // Set Input Node Values
                a.Run();                  // Forward Pass
                a.Set_Error(train_data[i]);  // Set Desired Output in
output layer
                a.Learn();                // Backward Pass
                if (fabs(a.Get_Value(0)-train_data[i]-
>Out(0))<tolerance)
                {
                    good++;
                }
                ScreenToFile << endl << " ID: " << train_data[i]-
>Get_ID() << "  Good " << good << "  opt0: " << a.Get_Value(0) << "
target: " << train_data[i]->Out(0);

                total_error+=fabs(a.Get_Error(0));
                    //if (total_error < tolerance2) good2++;
            }
            if (iteration%10==0)
            {
                ScreenToFile << iteration << ".   " << good
                          << "/" << NUM_IN_TRAIN_SET   << " Error: " <<
setprecision(15)
                          << total_error << endl;
                end = clock();
                timeTaken = (end - start)/CLK_TCK;
                Memo1->Lines->Add(IntToStr(iteration)+" iterations
completed. Time taken is " + FloatToStr(timeTaken)+"secs");
                Memo1->Lines->Add("   Number of trained patterns so far
is" + IntToStr(good));
            }
            iteration++;
            if (iteration > 200000) break;
    }// of while

     if (iteration <=200000)
     {
            // Save Backprop
        ofstream outfile("bp_vert.net");
        a.Save(outfile);
        outfile.close();
     }
     else
     {
        Memo1->Lines->Add("Training process failed after " +
IntToStr(iteration) + " iterations.");
     }

    end = clock();
    timeTaken = (end - start)/CLK_TCK;

    Memo1->Lines->Add("Training process finished after " +
IntToStr(iteration) + " iterations. Time taken was " +
FloatToStr(timeTaken)+"secs");
}
//-----------------------------------------------------------------------
void __fastcall TMainForm::Test1Click(TObject *Sender)
{
    clock_t start, end;
    start = clock();
    const int NUM_IN_TEST_SET = 16;
    const int DATA_WIDTH = 8;
    const int DATA_HEIGHT = 96;
    const int INPUT_SIZE = DATA_WIDTH*DATA_HEIGHT;   //stripe of width
8, height 96
    const int OUTPUT_SIZE =1;

    ifstream read_test("TestingSet_8x96In_1Out.txt");
    ofstream ScreenToFile("TestingFile.txt",ios::app);


      // Create Training Set - XOR problem
    Pattern *test_data[NUM_IN_TEST_SET];
    for (int i = 0; i < NUM_IN_TEST_SET; i++)
    {
     test_data[i]=new Pattern(INPUT_SIZE, OUTPUT_SIZE, read_test );
    }

// Create New Backprop Network
     Backprop_Network b;

// Load Backprop
     ifstream infile("bp_vert.net");
     b.Load(infile);
     infile.close();

// Run Backprop
     for (int i=0; i<NUM_IN_TEST_SET; i++)
          {
          b.Set_Value(test_data[i]);                // Set Input Node
Values

          b.Run();                                  // Forward Pass

          ScreenToFile << "\nID: " << test_data[i]->Get_ID() <<
"\nBackprop: (";
            for (int j = 0; j< OUTPUT_SIZE; j++)
            {
                 ScreenToFile << b.Get_Value(j);
                 if (j != OUTPUT_SIZE-1) ScreenToFile << ",";
            }
            ScreenToFile << ")" << endl;

          ScreenToFile << "Backprop: (";
            for (int j = 0; j< OUTPUT_SIZE; j++)
            {
                 ScreenToFile << (int)(b.Get_Value(j)+0.5);
                 if (j != OUTPUT_SIZE-1) ScreenToFile << ",";
            }
            ScreenToFile << ")" << endl;

          ScreenToFile << "  Actual: (";
          for (int j = 0; j< OUTPUT_SIZE; j++)
            {
              ScreenToFile << test_data[i]->Out(j);
              if (j!= OUTPUT_SIZE-1) ScreenToFile << ",";
            }
            ScreenToFile  << ")" << endl;

          }

    end = clock();
    float timeTaken = (end - start)/CLK_TCK;

    Memo1->Lines->Add("Testing finished after " +
FloatToStr(timeTaken)+"secs");
}
//-----------------------------------------------------------------------
void __fastcall TMainForm::CategoriseLinesFound1Click(TObject *Sender)
{
    int *theta, *rho;
    int *type;
    int numOfLines;

    ifstream readLines("BFPeaks_5x5WithSubLines.txt");
    ofstream writeLineCategories("LineCategories.txt");

    readLines >> numOfLines;
    theta = new int[numOfLines];
    rho = new int[numOfLines];

    type = new int[numOfLines];

    int typeCount[8];    // 9th category for uncategories lines
    for (int j = 0; j<8; j++) typeCount[j] = 0;
    for (int i = 0; i < numOfLines; i++)
    {
                        //assigning type
        readLines >> theta[i] >> rho[i];
        if ((theta[i]>=-5)&&(theta[i] <= 4))
            {type[i] = 0; typeCount[0]++;} //vertical
        else if (theta[i] <= 24)
            {type[i] = 1; typeCount[1]++;} //vertical backslash
        else if (theta[i] <= 64)
            {type[i] = 2; typeCount[2]++;} //backslash
        else if (theta[i] <= 84)
            {type[i] = 3; typeCount[3]++;} //horizontal backslash
        else if (theta[i] <= 94)
            {type[i] = 4; typeCount[4]++;} //horizontal
        else if (theta[i] <= 114)
            {type[i] = 5; typeCount[5]++;} //horizontal slash
        else if (theta[i] <= 154)
            {type[i] = 6; typeCount[6]++;} //slash
        else if (theta[i] <= 174)
            {type[i] = 7; typeCount[7]++;} //vertical slash
        else if (theta[i] <= 185)
            {type[i] = 0; typeCount[0]++;} //vertical

        writeLineCategories << theta[i] << " " << rho[i] << " "<<
type[i] << endl;
    }

    writeLineCategories << numOfLines << endl;
    for (int j = 0; j< 8; j++) //all types
    {
        writeLineCategories << endl << typeCount[j] << "\t";
        Memo1->Lines->Add("Number of lines in category " + IntToStr(j) +
" is " + IntToStr(typeCount[j]));
    }
}
//---------------------------------------------------------------------
void __fastcall TMainForm::PlotSubLinesColoringaccordingtolines1Click(
      TObject *Sender)
{
    ifstream readPoints("Associated Lines.txt");
    int numOfLines, numOfPoints, pointID;
    int x,y,i,j;
    bool moreLines = true;
    int numOfSubLines = 0;

    while (moreLines)
    {
        if (readPoints >> numOfLines)
        {
            j = 0;
           while (j < numOfLines)
             {
                readPoints >> numOfPoints;
                i = 0;
                while (i <  numOfPoints)
                {
                    readPoints >> pointID;
                    x = pointID%128;
                    y = pointID/128;
                    switch ( numOfSubLines%12 ) // j is count of lines
                    {
                    case 0 : Image3->Canvas->Pixels[x][y] = clRed;
break;
                    case 1 : Image3->Canvas->Pixels[x][y] = clBlue;
break;
                    case 2 : Image3->Canvas->Pixels[x][y] = clGreen;
break;
                    case 3 : Image3->Canvas->Pixels[x][y] = clTeal;
break;
                    case 4 : Image3->Canvas->Pixels[x][y] = clMaroon;
break;
                    case 5 : Image3->Canvas->Pixels[x][y] = clNavy;
break;
                    case 6 : Image3->Canvas->Pixels[x][y] = clLime;
break;
                    case 7 : Image3->Canvas->Pixels[x][y] = clDkGray;
break;
                    case 8 : Image3->Canvas->Pixels[x][y] = clAqua;
break;
                    case 9 : Image3->Canvas->Pixels[x][y] = clFuchsia;
break;
                    case 10 : Image3->Canvas->Pixels[x][y]= clYellow;
break;
                    case 11 : Image3->Canvas->Pixels[x][y]= clPurple;
break;
                    case 12 : Image3->Canvas->Pixels[x][y]= clOlive;
break;
                    }
                    i++;
                }
                j++;
                numOfSubLines++;
            }
        }
        else moreLines = false;
    }
}
//---------------------------------------------------------------------
void __fastcall TMainForm::BreakdowntoSubImages1Click(TObject *Sender)
{
    const int IMAGE_WIDTH = 128;
    const int IMAGE_HEIGHT = 96;
    const int SUB_IMAGE_WIDTH=8;
    const int SUB_IMAGE_HEIGHT=8;
    //const int NUM_OF_SUBS =
IMAGE_WIDTH*IMAGE_HEIGHT/(SUB_IMAGE_WIDTH*SUB_IMAGE_HEIGHT);
    double image[IMAGE_WIDTH][IMAGE_HEIGHT];
    ifstream read_image("thinnedImage.txt");
    ofstream writeSubImage("TestingSet_8x8In_1Out.txt");


    for (int j = 0; j<IMAGE_HEIGHT; j++)
    {
        for (int i = 0; i < IMAGE_WIDTH; i++)
        {

            read_image >> image[i][j];
            //writeSubImage << image[i][j] << " ";
        }
        //writeSubImage << endl;
    }
    int subImageCounter = 0;
    //int slideIncrement = SUB_IMAGE_WIDTH;
    double iArray[SUB_IMAGE_WIDTH*SUB_IMAGE_HEIGHT];
    double oArray[1]; oArray[0] = 7.0;
    int subImIndex;

    for (int j = 0; j<IMAGE_HEIGHT; j=j+SUB_IMAGE_HEIGHT)
    {
        for (int i = 0; i < IMAGE_WIDTH; i=i+SUB_IMAGE_WIDTH)
        {
            subImIndex = 0;
            for (int n = j; n < j+SUB_IMAGE_HEIGHT; n++)   //start
building a sub-image
            {
                for (int m = i; m < i + SUB_IMAGE_WIDTH; m++)
                {
                    iArray[subImIndex] = image[m][n];
                    subImIndex++;
                }
            }
                                  //end of building a sub-image. set up
pattern
            Pattern p( SUB_IMAGE_WIDTH*SUB_IMAGE_HEIGHT, 1,
subImageCounter,
                  iArray, oArray );
            p.PrintToFile(writeSubImage, SUB_IMAGE_WIDTH);
            subImageCounter++;
        }
    }
}
//----------------------------------------------------------------------
void __fastcall TMainForm::GenerateTrainingSet1Click(TObject *Sender)
{
    clock_t start, end;
    start = clock();
    float timeTaken;

    srand(123);
    const int NUM_IN_TRAIN_SET = 2;
    const int DATA_WIDTH = 8;           //num of columns in sub-image
results
    const int DATA_HEIGHT = 192;        //num of sub-images
    const int INPUT_SIZE = DATA_WIDTH*DATA_HEIGHT;   //stripe of width
8, height 96
    const int OUTPUT_SIZE = 16;        //width of image, showing
vanishing point
    ifstream read_train("TrainingSet_FullPicture8x192In_16Out.txt");
    ofstream ScreenToFile("TrainingFile_FullPicture.txt");

      // Create Training Set
    Pattern *train_data[NUM_IN_TRAIN_SET];

                                // sizes  id    input     output
                            // -----  --    -----     ------
    for (int i = 0; i < NUM_IN_TRAIN_SET; i++)
    {
        train_data[i]=new Pattern(INPUT_SIZE, OUTPUT_SIZE, read_train );
    }

    for (int i = 0; i< NUM_IN_TRAIN_SET; i++) train_data[i]-
>PrintToFile(ScreenToFile, DATA_WIDTH);

     // Create Backprop Network
     Backprop_Network a(0.45, 0.9,  3,   INPUT_SIZE,4*192,OUTPUT_SIZE);
// 3 layers, INPUT_SIZE inputs, 3 middle, OUTPUT_SIZE output
                                         // Learning rate 0.45, momemtum
0.9
// Train Backprop Network
     long iteration=0;
     int good=0;
     //int good2=0;  //to be based on total error rather than error on
each output
     double tolerance=0.5;
     //double tolerance2 = 0.5;
     double total_error;
     bool condition;

    while (good<NUM_IN_TRAIN_SET)   // Train until all patterns are
correct
    {
        good=0;
        total_error=0.0;

        for (int i=0; i<NUM_IN_TRAIN_SET; i++)
            {
                a.Set_Value(train_data[i]);  // Set Input Node Values
                a.Run();                  // Forward Pass
                a.Set_Error(train_data[i]);  // Set Desired Output in
output layer
                a.Learn();                // Backward Pass

                condition = true;
                for (int k = 0; k < OUTPUT_SIZE; k++)
                {
                    condition = condition&&(fabs(a.Get_Value(k)-
train_data[i]->Out(k))<tolerance);
                    ScreenToFile << "\nOutput num " << k << " is "  <<
a.Get_Value(k) << " cond is " << (fabs(a.Get_Value(k)-train_data[i]-
>Out(k))<tolerance);
                    total_error+=a.Get_Value(k)-train_data[i]->Out(k);
                }

                if (condition)
                {
                    good++;
                }
            }

            if (iteration%20==0)
            {
                ScreenToFile << iteration << ".   " << good
                          << "/" << NUM_IN_TRAIN_SET   << " Error: " <<
setprecision(15)
                          << total_error << endl;
                end = clock();
                timeTaken = (end - start)/CLK_TCK;
                Memo1->Lines->Add(IntToStr(iteration)+" iterations
completed. Time taken is " + FloatToStr(timeTaken)+"secs");
                Memo1->Lines->Add("   Number of trained patterns so far
is" + IntToStr(good));
            }
            iteration++;
            if (iteration > 200000) break;
    }// of while

     if (iteration <=200000)
     {
            // Save Backprop
        ofstream outfile("bp_FP8x192In_16Out.net");
        a.Save(outfile);
        outfile.close();
     }
     else
     {
        Memo1->Lines->Add("Training process failed after " +
IntToStr(iteration) + " iterations.");
     }

    end = clock();
    timeTaken = (end - start)/CLK_TCK;

    Memo1->Lines->Add("Training process finished after " +
IntToStr(iteration) + " iterations. Time taken was " +
FloatToStr(timeTaken)+"secs");
}
//---------------------------------------------------------------------
void __fastcall TMainForm::HT1Click(TObject *Sender)
{
    TObject *s;
    PrintTestResults1Click(s); //check for lines with NNs

    //ofstream writeRes("Sub Image Results.txt");

    int catCode = StrToInt(InputBox("Category", "Enter code for
category; 8 for all","8"));

    clock_t start, end;
    start = clock();

    I.Hybrid14HT(sResults[0],catCode);
    I.h14buildContributingPointsArray(sResults[0],catCode);

    end = clock();
    float timeTaken = (end - start)/CLK_TCK;

    Memo1->Lines->Add("Hybrid 14 Hough transform completed in " +
FloatToStr(timeTaken)+"secs");

    I.h14printAccArray();
}
//---------------------------------------------------------------------
void __fastcall TMainForm::ShowResults1Click(TObject *Sender)
{
    for (int subID = 0; subID < 192; subID++)
    {
        if (sResults[0].getVer(subID)==1)
        {
            plotFromPolar_8x8NNSub(0, 0, subID);
        }
    }
}
//---------------------------------------------------------------------

void __fastcall TMainForm::FormSublines1Click(TObject *Sender)
{

    clock_t start, end;
    start = clock();

    I.h14formSubLines();
    end = clock();
    float timeTaken = (end - start)/CLK_TCK;

    Memo1->Lines->Add("Sub lines determined in " +
FloatToStr(timeTaken)+"secs");
    Memo1->Lines->Add("   - number of lines with valid sublines is " +
IntToStr(I.getNumOfLinesWithSubLines()));
    Memo1->Lines->Add("   - number of sublines is " +
IntToStr(I.getNumOfSubLines()));

    I.h14printBFPeaksWithSubLines();

}
//---------------------------------------------------------------------------
void __fastcall TMainForm::FindEndpoints1Click(TObject *Sender)
{
   ifstream readPoints("Associated Lines.txt");
   ifstream readGrad("Gradient_Intercept.txt");
   ofstream writeLineParameters("EndPoints.txt");
    int numOfLines, numOfPoints, pointID;
    int x,y,i,j;
    bool moreLines = true;
    int numOfSubLines = 0;

    int dist, maxDist, minDist;
    int maxDistIndex, maxDistX, maxDistY, maxDisX, maxDisY;
    int minDistIndex, minDistX, minDistY, minDisX, minDisY;
    int x_pivot, y_pivot, length, parentLineNum=-1;
    float m, dumC; int dumNum;//gradient

    readGrad >> dumNum;   //do a dummy read of number of lines at the
begining of file

    while (moreLines)
    {
        if (readPoints >> numOfLines)
        {

            j = 0;
           while (j < numOfLines)
           {
                //readGradCheck--;
                if (j == 0)
                {
                    readGrad >> m >> dumC;
                    parentLineNum++;
                }
                readPoints >> numOfPoints;
                i = 0;
                maxDist = 0;
                minDist = 0;
                while (i <  numOfPoints)
                {
                    readPoints >> pointID;
                    x = pointID%128;
                    y = pointID/128;
                    if (i == 0)
                    {
                        x_pivot = x;
                        y_pivot =y;
                        maxDistIndex = pointID;
                        minDistIndex = pointID;
                    }
                    if (abs(m) <=1) dist = (x_pivot -x);
                    else dist = (y_pivot - y);

                    if (dist > maxDist)
                    {
                        maxDist = dist;
                        maxDistIndex = pointID;
                    }
                    if (dist < minDist)
                    {
                        minDist = dist;
                        minDistIndex = pointID;
                    }
                    i++;
                } //end of while i < numOfPoints
                maxDisX = maxDistIndex%128;
                maxDisY = maxDistIndex/128;
                minDisX = minDistIndex%128;
                minDisY = minDistIndex/128;

                maxDistX = maxDistIndex%128 - 64;
                maxDistY = -1*(maxDistIndex/128-47);
                minDistX = minDistIndex%128 - 64;
                minDistY = -1*(minDistIndex/128-47);

                length =  sqrt((maxDisX-minDisX)*(maxDisX-
minDisX)+(maxDisY-minDisY)*(maxDisY-minDisY));
                writeLineParameters << "\n" << parentLineNum << "\t" <<
maxDistX << " " << maxDistY << "\t" << minDistX
                                          << " " << minDistY << "\t" <<
length;// << "\tGrad: " << m;
                j++;
                numOfSubLines++;
            }//end of while j < num of lines
        }//end of if not end of file
        else moreLines = false;
    }//end of while morelines
}
//----------------------------------------------------------------------


void __fastcall TMainForm::Finddistancetoendpoints1Click(TObject
*Sender)
{
    int x, y;    //VP
    int *theta, *rho;
    int *xInt, *yInt, *x1, *y1, *x2, *y2;
    int *type;
    float *distanceFromVP;
    int dumVal;
    int numOfLines;
    float a, b, c, distance, cosC, C;

    ifstream readLines("ActualSignificants.txt");
    ifstream readEndPoints("EndPoints.txt");
    ofstream writeLineCategories("DistToVP.txt");
    ifstream readVP("VP.txt");

    readVP >> x >> y;
    readLines >> numOfLines;

    theta = new int[numOfLines];
    rho = new int[numOfLines];
    xInt = new int[numOfLines];
    yInt = new int[numOfLines];
    x1 = new int[numOfLines];
    y1 = new int[numOfLines];
    x2 = new int[numOfLines];
    y2 = new int[numOfLines];
    type = new int[numOfLines];
    distanceFromVP = new float[numOfLines];
    //float rhoCurve, cosT, sinT;
    //int rhoCurve_int;
    int typeCount[8];

    for (int i = 0; i < numOfLines; i++)
    {
                        //assigning type
        readLines >> theta[i] >> rho[i] >> dumVal >> x1[i] >> y1[i] >>
x2[i] >> y2[i];
        readEndPoints >> dumVal >> x1[i] >> y1[i] >> x2[i] >> y2[i] >>
a;


                        //assigning distance
        b = sqrt((x1[i] - x)*(x1[i] - x) + (y1[i] - y)*(y1[i]- y));
//distance b/w VP and NE
        a = sqrt((x1[i] - x2[i])*(x1[i] - x2[i]) + (y1[i] -
y2[i])*(y1[i]- y2[i]));  //distance b/w SW and NE
        c = sqrt((x2[i] - x)*(x2[i] - x) + (y2[i] - y)*(y2[i]- y));
//distance b/w VP and SW

                        //find angle between a and b with cosine formula
        cosC = (a*a + b*b - c*c)/(2*a*b);
        C = acos(cosC);

                        // find distance using sine formula
        distance = a * sin(C);
        distanceFromVP[i] = distance;

        writeLineCategories << distanceFromVP[i] << endl;
    }
}
//---------------------------------------------------------------------------
void __fastcall TMainForm::FinddistancestoVP1Click(TObject *Sender)
{
    int x, y, x_intn, y_intn, theta, rho, theta2, rho2, numOfLines;
    float m, c, m2, c2, distance;
    ifstream readVP("VP.txt");
    ifstream readLines("BFPeaks_5x5WithSubLines.txt");
    ofstream writeDistToVP("DistToVP.txt");

    float rhoCurve;

    readVP >> x >> y;
    readLines >> numOfLines;
    for (int i = 0; i < numOfLines; i++)
    {
            //read VP, theta, rho
        readLines >> theta >> rho;


            //find theta and rho for perpendicular line
        theta2 = theta + 90;
        rho2 = x*cos(theta2*glb_CONVERT) + y*sin(theta2*glb_CONVERT);

            // convert to m, c, m', c'
        switch (theta)
        {
```

```
        case 90:  //i.e. 90: horizontal line
            m = 0;
            c = rho;
            m2 = 99999;
            c2 = 99999;
            x_intn = x;
            y_intn = c;
        break;

        case 180:
            m = 99999;
            c = 99999;
            //m2 = 0;
            //c2 = -1*rho;
            x_intn = -1*rho;
            y_intn = y;
        break;

        case 0:          //vertical line
            m = 99999;
            c = 99999;
            //m2 = 0;
            //c2 = rho;
            x_intn = rho;
            y_intn = y;
        break;

        default:

            m = -1/(tan(theta*glb_CONVERT));
            c = rho/(sin(theta*glb_CONVERT));
            m2 = -1/(tan(theta2*glb_CONVERT));
            c2 = rho2/(sin(theta2*glb_CONVERT));
            x_intn = -1*((c-c2)/(m-m2));
            y_intn = m2*x_intn + c2;

        } // end of switch

            //find intersection point
        distance =  sqrt((x_intn - x)*(x_intn - x) + (y_intn - 
y)*(y_intn - y));

            //determine sign of distance
        rhoCurve = x*cos(theta*glb_CONVERT) + y*sin(theta*glb_CONVERT);
        if (rhoCurve <= (float)rho) distance = -1*distance;

        writeDistToVP << endl /*<< "\t" << x_intn << "\t" << y_intn << 
"\t"*/ << distance;
    }

}
//---------------------------------------------------------------------
void __fastcall TMainForm::LeftCorridorEdge1Click(TObject *Sender)
{
    int x, y, numOfLines, *theta, *rho, countOfInRange, minSWIndex, 
minVPIndex;
    int *x1,*y1,*x2,*y2, *score, dumVal, *category;
    float distToSW, *distToVP, minSW, minVP;
    bool *inRange, *assignedSW, *assignedVP;
    int numOfSubLines;
    ifstream readVP("VP.txt");

    ifstream readLines("BFPeaks_5x5WithSubLines.txt");
    ifstream readDistToVP("DistToVP.txt");
    ifstream readEndPoints("EndPoints.txt");
    ifstream readNumOfSubLines("NumOfSubLines.txt");
    ifstream readCategories("LineCategories.txt");
    ofstream writeCorridor("Corridor_LE.txt");

    readNumOfSubLines >> numOfSubLines;
    readLines >> numOfLines;
    readVP >> x, y;

    theta = new int[numOfLines];
    rho = new int[numOfLines];
    inRange = new bool[numOfLines];
    distToVP = new float[numOfLines];
    category = new int[numOfLines];

    assignedSW = new bool[numOfSubLines];
    assignedVP = new bool[numOfSubLines];
    score = new int[numOfSubLines];
    x1 = new int[numOfSubLines];
    y1 = new int[numOfSubLines];
    x2 = new int[numOfSubLines];
    y2 = new int[numOfSubLines];

    for (int i = 0; i<numOfLines; i++)
    {
        readLines >> theta[i] >> rho[i];
        if ((theta[i]>=90)&&(theta[i]<180))
        {
            inRange[i] = true;
            countOfInRange++;
        }
        else inRange[i] = false;
        readDistToVP >> distToVP[i];
        readCategories >> dumVal >> dumVal >> category[i];
    }

    int *parentLineNum = new int[numOfSubLines];
    for (int i = 0; i<numOfSubLines; i++)
    {
        readEndPoints >> parentLineNum[i];
        readEndPoints >>  x1[i] >> y1[i] >> x2[i] >> y2[i] >> dumVal;
    }

    for (int i = 0; i < numOfSubLines; i++)
    {
        score[i]=0;
        assignedSW[i]=false;
        assignedVP[i]=false;
    }

    int parentNum;
    float disToVP;
    for (int j = 5; j>0; j--)
    {
        minSWIndex = -1; minSW = 99999;
        minVPIndex = -1; minVP = 99999;
        for (int i = 0; i<numOfSubLines; i++)
        {
            parentNum = parentLineNum[i];
            if (inRange[parentNum] == true)
            {
                if (assignedSW[i]==false)
                {
                    distToSW = findDisToSW(x1[i], y1[i], x2[i], y2[i]);
```

```
                    if (distToSW <= minSW)
                    {
                        minSW = distToSW;
                        minSWIndex = i;
                    }
                }
                if (assignedVP[i]==false)
                {
                    disToVP = fabs(distToVP[parentLineNum[i]]);
                    if (disToVP <= minVP)
                    {
                        minVP = fabs(distToVP[parentLineNum[i]]);
                        minVPIndex = i;
                    }
                }
            }
        }

        assignedSW[minSWIndex]=true;
        score[minSWIndex] = score[minSWIndex]+j;

        assignedVP[minVPIndex]=true;
        score[minVPIndex] = score[minVPIndex]+j;


    }

    for (int i = 0; i<numOfSubLines; i++)
    {
        if (inRange[parentLineNum[i]] == true)
            switch (category[parentLineNum[i]])
            {
                case 5:
                case 6: score[i] = score[i]+7; break;
                case 4: score[i] = score[i]+5; break;
                case 7: score[i] = score[i]+3; break;
                case 0: score[i] = score[i]+2; break;
            }
    }
            //find line with the highest score
    int max = 0, maxIndex = -1;
    for (int i = 0; i < numOfSubLines; i++)
    {
        if (score[i] > max)
        {
            max = score[i];
            maxIndex = i;
        }
    }

    writeCorridor << x1[maxIndex] << "\t" << y1[maxIndex] << "\t" <<
x2[maxIndex] << "\t" << y2[maxIndex] << "\t" << parentLineNum[maxIndex];
    int x_1 = x1[maxIndex]+64;
    int y_1 = 47-y1[maxIndex];
    int x_2 = x2[maxIndex]+64;
    int y_2 = 47-y2[maxIndex];

    Image3->Canvas->Pen->Color = clRed;
    Image3->Canvas->MoveTo(x_1, y_1);
    Image3->Canvas->LineTo(x_2, y_2);
    Image3->Canvas->Pixels[x_2][y_2] = clRed;

}
//---------------------------------------------------------------------
```

```
void __fastcall TMainForm::Gettopedge1Click(TObject *Sender)
{
    int x, y, numOfLines, *theta, dumRho,  minVPIndex, minMPIndex;
    int *x1,*y1,*x2,*y2, *score, dumVal, *category;
    float *distToVP, minSW, minVP;
    bool *assignedVP, *assignedMP;
    int numOfSubLines;
    ifstream readVP("VP.txt");

    ifstream readLines("BFPeaks_5x5WithSubLines.txt");
    ifstream readDistToVP("DistToVP.txt");
    ifstream readEndPoints("EndPoints.txt");
    ifstream readNumOfSubLines("NumOfSubLines.txt");
    ifstream readCategories("LineCategories.txt");
    ofstream writeDoor("Door_TE.txt");

    readNumOfSubLines >> numOfSubLines;
    readLines >> numOfLines;
    readVP >> x >> y;

    theta = new int[numOfLines];
    distToVP = new float[numOfLines];
    category = new int[numOfLines];

    assignedVP = new bool[numOfSubLines];
    assignedMP = new bool[numOfSubLines];
    score = new int[numOfSubLines];
    x1 = new int[numOfSubLines];
    y1 = new int[numOfSubLines];
    x2 = new int[numOfSubLines];
    y2 = new int[numOfSubLines];

    for (int i = 0; i<numOfLines; i++)
    {
        readLines >> theta[i] >> dumRho;
        readDistToVP >> distToVP[i];
        readCategories >> dumVal >> dumVal >> category[i];
    }

    int *parentLineNum = new int[numOfSubLines];
    for (int i = 0; i<numOfSubLines; i++)
    {
        readEndPoints >> parentLineNum[i];
        readEndPoints >>  x1[i] >> y1[i] >> x2[i] >> y2[i] >> dumVal;
    }

    for (int i = 0; i < numOfSubLines; i++)
    {
        score[i]=0;
        assignedVP[i]=false;
        assignedMP[i]=false;
    }

    int parentNum;
    float disToVP;
    for (int j = 5; j>0; j--)
    {
        minVPIndex = -1; minVP = 99999;
        for (int i = 0; i<numOfSubLines; i++)
        {
            parentNum = parentLineNum[i];
            if (category[parentNum] == 4)
            {
                if (assignedVP[i]==false)
                {
                    disToVP = distToVP[parentNum];
                    if (disToVP < 0) //place on right side of VP
                        if (fabs(disToVP) <= minVP)
                        {
                            minVP =
abs(disToVP);//fabs(distToVP[parentLineNum[i]]);
                            minVPIndex = i;

                        }
                }

            }
        }
        assignedVP[minVPIndex]=true;
        score[minVPIndex] = score[minVPIndex]+j;
    }

            //find line with the highest score
    int max = 0, maxIndex = -1;
    for (int i = 0; i < numOfSubLines; i++)
    {
        if (score[i] > max)
        {
            max = score[i];
            maxIndex = i;
        }
    }
    writeDoor << x1[maxIndex] << "\t" << y1[maxIndex] << "\t" <<
x2[maxIndex] << "\t" << y2[maxIndex] << "\t" << parentLineNum[maxIndex];

    int x_1 = x1[maxIndex]+64;
    int y_1 = 47-y1[maxIndex];
    int x_2 = x2[maxIndex]+64;
    int y_2 = 47-y2[maxIndex];

    Image3->Canvas->Pen->Color = clRed;
    Image3->Canvas->MoveTo(x_1, y_1);
    Image3->Canvas->LineTo(x_2, y_2);
    Image3->Canvas->Pixels[x_2][y_2] = clRed;
}
//-----------------------------------------------------------------------
```

Appendix F Technical Specifications of the Koala Robot

Elements	Technical Information
Processor	Motorola 68331 @ 22MHz
RAM	1Mbyte
ROM	1Mbyte
Motion	2 DC brushed servo motors with integrated incremental encoders (roughly 19 pulses per mm of robot motion)
Speed	Max: 0.4 m/s Min: 5 mm/s
Sensors	16 Infra-red proximity and ambient light sensors 4 optional triangulation longer-range IR sensors Up to 6 optional ultrasonic sonar sensors Battery and ambient temperature Motor torque and global power consumption
Power	Rechargeable NiMH Battery with charge level memory The battery pack can be easily removed and replaced.
Clearance	Max: 30mm
Autonomy (4Ah battery)	Approx 6 hours (moving continuously without payload) Approx 4 hours (moving continuously with maximum payload)
Extension Bus	Expansion modules can be added to the robot such as the Kameleon or PC104 boards. Khepera turrets (with local processor) are also supported using an adaptor turret. An accessory deck is provided at the front of the robot for mounting any other custom equipment you wish.
User Available I/O	12 digital inputs [5..12V] 4 CMOS / TTL digital outputs 8 power (open collector) digital outputs [12V 250mA/output] 6 analog inputs (10 bit A/D converter, 4.096v range)
Size	Length: 32 cm Width: 32 cm Height: 20 cm
Weight	4kg with battery 3.6kg with DC-DC converter
Maximum Payload	3kg

Source: http://www.cyberbotics.com/products/robots/koala.html

Appendix G Time Complexity Analysis for Some Processes in Hough Transform System

Conversion to gray scale

This process, as pointed out in *4.1.4 Intensity Determination*, the process of conversion of an image to gray scale extracts the levels of red, blue and green for each pixel in the image and averages them. This process therefore has $\mathrm{O} = \Omega = \Theta = n$.

Edge Detection

4.2.1 Edge Threshold determination pointed out that this process involves applying certain operators to every pixel in the image. This means therefore that $\mathrm{O} = \Omega = \Theta = n$ for this process, where n is the number of pixels in the image.

Edge Thinning

From the discussion in *4.3 Edge Thinning* and (Park and Chen 200), every pixel that is an edge needs to be examined for the possibility of needing further thinning processing. This means the process is bounded above by n, i.e. $\mathrm{O} = n$, where n is the number of edges from edge detection.

Hough Transform

The Hough transform, as discussed in *5.1.5 Transformation and Accumulation*, is performed for all edge pixels in the image. Therefore, $\mathrm{O} = \Omega = \Theta = n$ for this process.

Peak Detection

According to *5.1.6 Peak Detection*, peak detection in this work involves 3 major steps:

4. Determination and application of the most appropriate threshold for the image under consideration
5. Application of the butterfly filter to entries above the threshold from step 1 above only
6. Determining which elements of the accumulator array selected from 2 above are local maxima within a 5 x 5 neighbourhood

Step 1 considers all elements of the accumulator array. Steps 2 and 3 will consider much fewer elements than all the elements of the accumulator array, but will still be of the order n, n being the number of element of the array. Therefore $\mathrm{O} = n$ for the peak detection process.

Determination of Sub-Lines

From the discussion in *5.2.1 Determination of Actual Lines*, determination of sub-lines involves two major steps:

1. Determine which points contributed to each line
2. Find the number of points on each line, noting which ones have $L_{\min}$ points or more, and are at least $S_{\min}$ pixels away from other points or lines.

For step 1, $\mathrm{O} = n^2$ as can be determined by studying the algorithm described in *5.2.1.2 Assigning Contributing Points to Sub-lines*. The core of the process has two loops, one nested in the other. Here n is the total number of points that are found to have made up the line under consideration, and whose sub-lines are to be determined.

For step 2, $\mathrm{O} = n$ as it primarily involves counting points in sub-images once.

For the complete process of determination of sub-lines therefore, $\mathrm{O} = n^2$.