

Copyright is owned by the Author of the thesis. Permission is given for a copy to be downloaded by an individual for the purpose of research and private study only. The thesis may not be reproduced elsewhere without the permission of the Author.

Development of an In-field Tree Imaging System

**A thesis
presented in partial fulfilment
of the requirements for the degree
of
Master of Technology
at
Massey University**

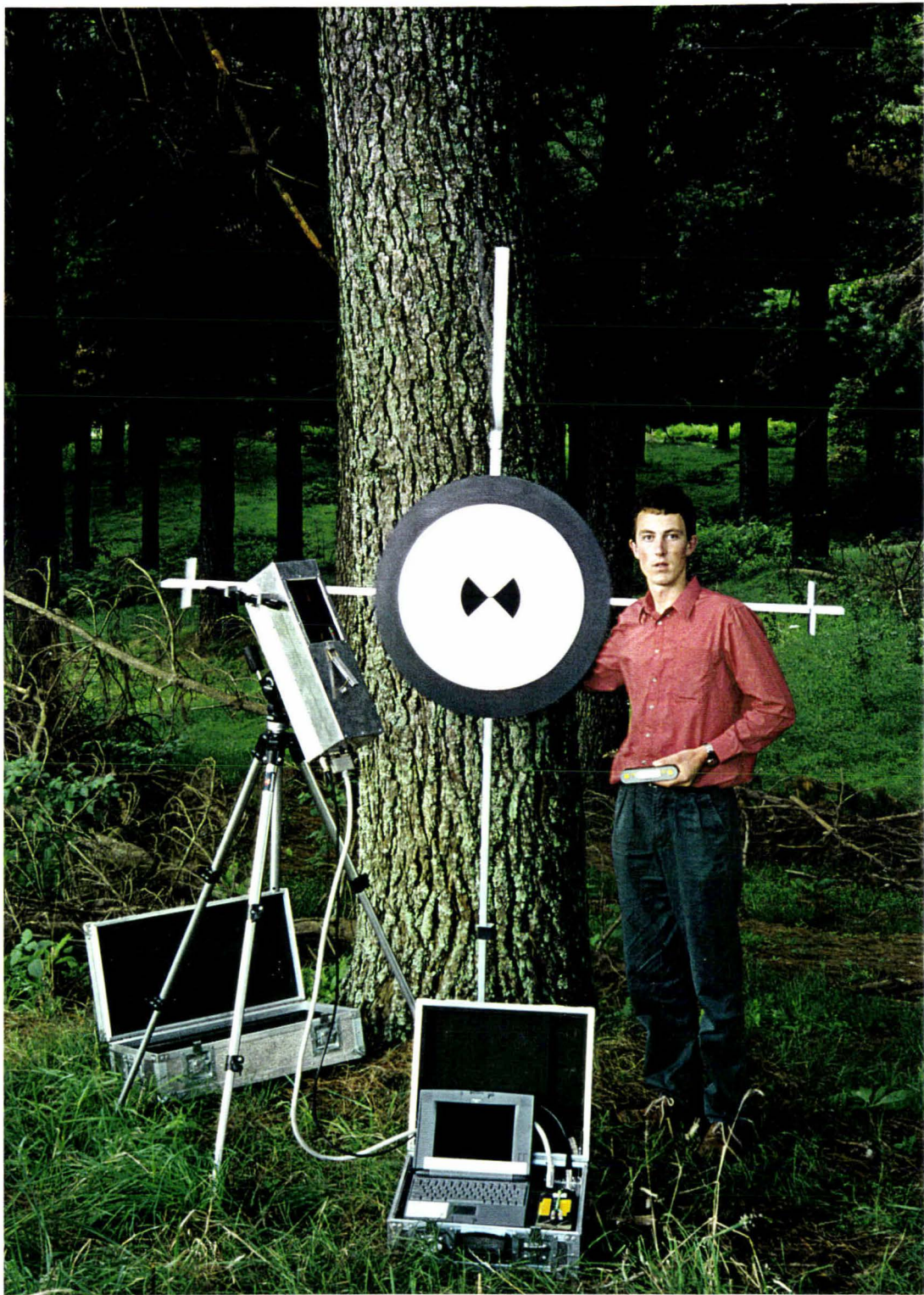
**By
Marijn Weehuizen
1996**



MASSEY UNIVERSITY



1061638784



Abstract

Quality inventory information is essential for optimal resource utilisation in the forestry industry. In-field tree imaging is a method which has been proposed to improve the preharvest inventory assessment of standing trees. It involves the application of digital imaging technology to this task. The method described generates a three dimensional model of each tree through the capture of two orthogonal images from ground level.

The images are captured and analysed using the "TreeScan" in-field tree imaging system. This thesis describes the design, development, and evaluation of the TreeScan system. The thesis can also be used as a technical reference for the system and as such contains appropriate technical and design detail.

The TreeScan system consists of a portable computer, a custom designed high resolution scanner with integral microcontroller, a calibration rod, and custom designed processing software. Images of trees are captured using the scanner which contains a CCD line scan camera and a precision scanning mechanism. Captured images are analysed on the portable computer using customised image processing software to estimate real world tree dimensions and shape.

The TreeScan system provides quantitative estimates of five tree parameters; height, sweep, stem diameter, branch diameter, and feature separation such as internodal distance. In addition to these estimates a three dimensional model is generated which can be further processed to determine the optimal stem breakdown into logs.

Table of Contents

Abstract	iii
Table of Contents	v
List of Figures	viii
List of Tables	xi
Publications	xi
Acknowledgements	xiii
Glossary	xvii
 CHAPTER 1 - INTRODUCTION AND BACKGROUND	 1
1.1 Scope of Research	2
1.2 Thesis Overview	2
1.3 Forest Industry Background	4
1.3.1 Introduction	4
1.3.2 Forestry and Sawmilling	5
1.3.3 Forest Operations	5
1.3.4 Inventory Assessment	6
1.4 Preharvest Inventory Assessment	7
1.4.1 MARVL Inventory Assessment	8
1.4.2 Weaknesses of the MARVL System	9
1.4.3 Possible Improvements	11
 CHAPTER 2 - IMPROVED FOREST STAND ASSESSMENT	
DESIGN PROPOSAL	13
2.1 In-field Tree Imaging to Improve Stand Assessment	14
2.1.1 Previous Research on Standing Tree Imaging	15
2.2 General Technology Options	16
2.2.1 Massey University Feasibility Study	16
2.2.2 Alternative Approach	18
2.3 Design Constraints	21
2.3.1 Constraints Imposed by Forest Conditions	21
2.3.2 Constraints Imposed by Technology Limitations	22
2.4 Proposed System	24
2.4.1 Improved Assessment Outline	24
2.4.2 Proposed Image Capture System overview	24
2.5 Design specification	26
 CHAPTER 3 - TREESCAN DESIGN CONSIDERATIONS	
AND THEORETICAL FOUNDATIONS	27
3.1 Design Overview	28
3.1.1 Systems Integration Project	29
3.2 TreeScan Operating Principle	31
3.2.1 TreeScan Estimates	33
3.3 Image Capture	35
3.3.1 Digital Image Capture	35
3.3.2 Primary Imaging Considerations	37

3.3.3	Area Sensor vs. Line Scan Build-up	38
3.3.4	Optical Considerations	41
3.3.5	Image Focus	45
3.4	Image Transfer and Storage	46
3.4.1	Scanner Interface	46
3.4.2	Image Storage	47
3.5	Parameter Extraction	48
3.5.1	Image Calibration	49
3.5.2	Planar Transformation Distortion Correction	51
3.5.3	Geometric Distortion Correction	53
3.6	Three Dimensional Model Construction	57
3.7	Implications of Image Capture Geometry	58
3.7.1	Tree Plane Variation	61
3.7.2	Calibration Alignment Variation	62
3.7.3	Image Processing and Feature Marking Precision	64
CHAPTER 4 - TREESCAN HARDWARE		67
4.1	TreeScan Hardware Overview	68
4.2	Scanner Hardware Overview	70
4.2.1	Scanner Controller Board	74
4.3	Microcontroller Subsystem	75
4.3.1	Microcontroller Subsystem Memory Organisation	77
4.3.2	Microcontroller Subsystem Memory Timing	78
4.4	SCSI Subsystem	80
4.4.1	Implementing SCSI : Design Specifications	82
4.4.2	SCSI Bus Controller (SN75C091A)	85
4.4.3	SCSI Subsystem Development Obstacles	86
4.5	Line Scan Camera Subsystem	87
4.5.1	Imaging Sensor Spectral Response	90
4.5.2	Line Scan Camera Subsystem Signal Timing	90
4.6	Additional Hardware	92
4.6.1	Scanning Mirror Subsystem	92
4.6.2	Lens Subsystem	95
4.6.3	Power Supply Subsystem	98
4.6.4	User Feedback	101
4.6.5	Scanner Chassis	102
4.6.6	Carrying Cases	103
4.7	Hardware Development Environment	103
CHAPTER 5 - TREESCAN SOFTWARE		105
5.1	TreeScan Software Overview	106
5.2	Image Capture Software	108
5.2.1	Overview	108
5.2.2	Image Build-up Algorithm	111
5.2.3	Image Block Capture Algorithm (Microcontroller)	114
5.2.4	SCSI Transfer Algorithm	118
5.2.5	Focus Algorithms	126
5.2.6	TreeScan Plug-in Software	129
5.2.7	Microcontroller Software	131
5.3	Tree parameter Extraction Software	132
5.3.1	Overview	132
5.3.2	Image Calibration	133
5.3.3	Feature Size Estimation in Two Dimensions	135
5.3.4	Three Dimensional Stem Shape Estimation	137

5.3.5	Possible Improvements to Parameter Extraction	139
5.3.6	TreeScan Macros	140
5.3.7	NIH Image Source Additions and Modifications	141
5.4	Software Development Environment	142
CHAPTER 6 - TREESCAN EVALUATION		143
6.1	Overview of Evaluation	144
6.2	Sequence of Evaluation Experiments	145
6.3	Hardware Calibration	147
6.3.1	Scanner Component Alignment	147
6.3.2	Measurement of Step Angle	149
6.4	TreeScan Characterisation	151
6.4.1	Image Capture Timing	151
6.4.2	TreeScan Resolution	153
6.4.3	Integration Time Adjustment	155
6.4.4	Focus Tests	156
6.5	Initial Accuracy Tests in Two Dimensions	157
6.6	Final Accuracy Tests in Two Dimensions	158
6.7	Accuracy Tests in Three Dimensions	160
CHAPTER 7 - FORESTRY IMPLICATIONS AND RECOMMENDATIONS		161
7.1	TreeScan Strengths and Limitations	162
7.2	Forestry Implications	166
7.3	Alternative Technology Uses	168
7.4	Future Work	169
CHAPTER 8 - SUMMARY		171
8.1	Summary	172
REFERENCES		175

Appendix A	Development Documentation for the TreeScan System
Appendix B	Sample Tree Analysis
Appendix C	Forestry Terms
Appendix D	Original TreeScan Project Proposal
Appendix E	System Error Calculations
Appendix F	TreeScan System Component List
Appendix G	TreeScan Schematics & Board Layout
Appendix H	Microcontroller Specifications and Memory Space Organisation
Appendix I	Additional SCSI Interface Specifications
Appendix J	SCSI Bus Controller Specifications
Appendix K	Macintosh SCSI Manager
Appendix L	SCSI Byte Loss Detection and Resend Scheme
Appendix M	Scanner Control Software
Appendix N	Image Processing Software

List of Figures

Figure 1.1	- Information loss inherent in the MARVL tree description	10
Figure 2.1	- Imaging technologies	17
Figure 2.2	- Effects of wind on captured images	19
Figure 2.3	- Alternative image capture approaches	20
Figure 2.4	- Improved forest stand assessment overview	24
Figure 2.5	- Image capture system principle	25
Figure 3.1	- Possible areas of technical difficulty	28
Figure 3.2	- TreeScan image capture	31
Figure 3.3	- Projection on a two dimensional plane	32
Figure 3.4	- TreeScan estimates	33
Figure 3.5	- Digital image capture	35
Figure 3.6	- CCD technology	36
Figure 3.7	- Photographic image capture distortion	38
Figure 3.8	- TreeScan image capture distortion	39
Figure 3.9	- Depth of field	42
Figure 3.10	- Modulation transfer function and relative illumination	44
Figure 3.11	- Definition of terms	49
Figure 3.12	- Simple perspective correction	51
Figure 3.13	- Two step perspective correction	52
Figure 3.14	- Geometric correction using derived O	53
Figure 3.15	- Correction based on calibration rod dimensions	54
Figure 3.16	- Distortion correction imprecision	55
Figure 3.17	- Measurement of angle O	56
Figure 3.18	- Three dimensional model generation	57
Figure 3.19	- Image capture geometry	59
Figure 3.20	- Tree plane variation	61
Figure 3.21	- Calibration alignment variation	62
Figure 4.1	- TreeScan system ready for image capture	69
Figure 4.2	- TreeScan scanner functional block diagram	70
Figure 4.3	- The scanner internal layout	71
Figure 4.4	- System signal flow diagram	73
Figure 4.5	- Scanner controller board layout	74
Figure 4.6	- Microcontroller block diagram schematic	76
Figure 4.7	- Microcontroller memory map	77
Figure 4.8	- EPROM read cycle timing	78
Figure 4.9	- SCSI block diagram schematic	81
Figure 4.10	- Typical command descriptor block	84
Figure 4.11	- Imaging sensor photosite layout	88
Figure 4.12	- LSC interface block diagram schematic	89
Figure 4.13	- CCD sensor spectral response	90
Figure 4.14	- Line scan camera timing	91

Figure 4.15 - Scanning mirror assembly	92
Figure 4.16 - Stepper motor controller block diagram schematic	94
Figure 4.17 - Mk1 and Mk2 lens systems	95
Figure 4.18 - Mk1 and Mk2 lens driving interface	97
Figure 4.19 - Power supply block diagram schematic	99
Figure 4.20 - Scanner chassis	102
Figure 4.21 - Hardware development environment	104
Figure 5.1 - Levels of TreeScan software	107
Figure 5.2 - Algorithms implemented in image capture software	108
Figure 5.3 - Image build-up sequence	111
Figure 5.4 - Image build-up algorithm	112
Figure 5.5 - Image build-up algorithm (description)	113
Figure 5.6 - Image block capture algorithm	114
Figure 5.7 - Image block capture algorithm (description)	115
Figure 5.8 - Line signal timing	116
Figure 5.9 - A/D signal timing	117
Figure 5.10 - A/D conversion (8 bit) microcontroller code	118
Figure 5.11 - Normal SCSI transfer	120
Figure 5.12 - Normal SCSI transfer (description)	121
Figure 5.13 - Image with byte loss problem	122
Figure 5.14 - Extended delays during SCSI transfer	124
Figure 5.15 - Byte loss detection and resend scheme	125
Figure 5.16 - Final autofocus algorithm	127
Figure 5.17 - TreeScan image capture user interface	129
Figure 5.18 - Parameter extraction sequence	132
Figure 5.19 - Marking of calibration points	134
Figure 5.20 - Two dimensional feature size estimates	135
Figure 5.21 - Generation of three dimensional stem model	137
Figure 3.22 - Sweep estimation from displayed tree model	138
Figure 5.23 - TreeScan processing and utility macros	123
Figure 6.1 - Distortion introduced by camera misalignment	147
Figure 6.2 - Camera alignment procedure	148
Figure 6.3 - Distortion introduced by mirror misalignment	149
Figure 6.4 - Image capture timing	152
Figure 6.5 - Image resolution effects	154
Figure 6.6 - Integration time adjustment	155
Figure 6.7 - Focus results	156
Figure 6.8 - Height errors with high imprecision	157
Figure 6.9 - Final accuracy tests in two dimensions	159

List of Tables

Table 2.1 - System design constraints	21
Table 3.1 - Standard f-numbers	41
Table 3.2 - Image acquisition time vs. data transfer rate	46
Table 3.3 - Scanner interface methods	47
Table 3.4 - Comparison of distortion correction methods	48
Table 3.5 - Sources of expected error in TreeScan	60
Table 3.6 - Height errors introduced by stem displacement	61
Table 3.7 - Width errors introduced by stem displacement	62
Table 3.8 - Errors introduced by variation in measured angle	63
Table 3.9 - Errors introduced by distance error	64
Table 4.1 - Availability of SCSI bus controllers	80
Table 4.2 - Line scan cameras available	87
Table 4.3 - Scanner power requirements	100
Table 6.1 - Measured pixel resolution	153
Table 7.1 - TreeScan strengths and limitations	163

Publications

The following publications were prepared during the research for this thesis:

- Weehuizen, M., Pugmire, R.H. (1994): The use of in-field tree imaging in the pre-harvest inventory assessment in the logging industry, Proceedings of New Zealand Postgraduate Conference for Engineering and Technology Students, Department of Production Technology, Massey University, 1994.
- Weehuizen, M., Pugmire, R.H. (1994): The use of in-field tree imaging in the pre-harvest inventory assessment in the logging industry, Proceedings of the Second New Zealand conference on Image Vision and Computing, Department of Production Technology, Massey University, 1994.

Acknowledgements

I would like to thank all the people who have been involved with the Tasman project over the last two years for making this research and development possible.

In particular I would like to thank my two supervisors Prof. Bob Hodgson and Dr. Ralph Pugmire for their valuable support and their vision in the project guidance. Thanks to Ralph Pugmire for his help during the many hours spent of poring over the 'unexplainable' development obstacles.

The contribution of Thomas Look was invaluable in the design and engineering of the mechanical components of the system.

Thanks also to the Gary Allen for the time spent constructing and testing the electronic aspects of the TreeScan system, and to Farshad Nourozi for his input into the design of the system.

I would also like to thank Tasman Forestry Ltd. for their backing of the development of the TreeScan system and continued support for further research. In particular I would like to thank Mike Colley (unfortunately moved on early in the project) for initiating the commitment of Tasman Forestry to in-field tree imaging.

Lastly, but certainly not least, I would like to thank Diana Foster for her editing skills on many chapter drafts and her unfaltering commitment whenever the project demanded more than its fair share of my time.

I must go down to the seas again, to the lonely sea and the sky,
And all I ask is a tall ship and a star to steer her by,
And the wheel's kick and the wind's song and the white sail's shaking,
And a grey mist on the sea's face and a grey dawn breaking.

I must go down to the seas again, for the call of the running tide
Is a wild call and a clear call that may not be denied;
And all I ask is a windy day with the white clouds flying,
And the flung spray and the blown spume, and the sea-gulls crying.

I must go down to the seas again, to the vagrant gypsy life,
To the gull's way and the whale's way where the wind's like a whetted
knife;
And all I ask is a merry yarn from a laughing fellow-rover,
And quiet sleep and a sweet dream when the long trick's over.

" Sea-Fever " by John Masefield

Glossary

A/D	Analog to Digital, used as A/D convertor.
CCD	Charge Coupled Device
CMOS	Complementary Metal Oxide Semiconductor
EPROM	Erasable/Programmable Read Only Memory
Kink	A short deflection of a log affecting less than 2 m of the log (see appendix C).
Log	A single section from a tree stem which has been cut into sections. A tree stem is cut into a number of logs for transport to the mill (typically 6-12 m in length).
Log grade	A measure of log quality and value. Each log grade has specifications which a log must meet (see appendix C).
LSC	Line Scan Camera
MARVL	Method of Assessment based on Recoverable Volume by Log type. The preharvest inventory system used by many forestry companies.
Plug-in	Macintosh code resource which complies with the Adobe interface specification and may be used to extend applications.
RAM	Random Access Memory
ROM	Read Only Memory
SCC	Scanner Control Command
SCSI	Small Computer Systems Interface, a high speed flexible computer interface commonly used to connect peripheral devices to computers.
SED	Small End Diameter, minimum diameter of a log.
Stem	A tree which has been felled but not yet cut into logs.
Sweep	Deviation from straightness along a length of log (see appendix C).
Wobble	Deviation from straightness of a log where the axis of a log deviates in two or more directions (see appendix C).
SBC	SCSI Bus Controller

Chapter 1

INTRODUCTION AND BACKGROUND

1.1	Scope of Research	2
1.2	Thesis Overview	2
1.3	Forest Industry Background	4
1.4	Preharvest Inventory Assessment	7

1.1 Scope of Research

The strategic objective of this research is to improve forest stand assessment by using imaging techniques to make the preharvest inventory assessment more quantitative. If successful this will have a far reaching impact on mensuration in the forestry industry.

In order to make the preharvest inventory assessment more quantitative two aspects are important; the dimensions of the standing radiata pine trees must be measured, and the method used to calculate recoverable volume from tree dimensions must be modified. The research for this masters project focuses on the development of a suitable image capture and processing system which can be used to accurately estimate tree dimensions.

More specifically, the objective of this masters project was to develop a line scan based image capture system that would allow the dimensions of standing pine trees to be estimated. As a result of this clearly defined objective this masterate has been a technology development project rather than a theoretical research project.

1.2 Thesis Overview

The research and development for this study takes the project from the design concept stage through the design and development stage up to the final testing stage. The structure of the thesis reflects this design path.

Chapter 1 provides an introduction and context for the research. The scope of the research is defined and a background to the forestry industry is provided with an emphasis on the preharvest inventory assessment. This chapter presents a statement of problem, independent of the proposed solution.

In chapter 2 alternative methods for improving inventory assessment are reviewed. The approaches identified in a Massey University feasibility study are outlined and analysed for design constraints. Based on this analysis a design proposal is put forward which provides the basis of the subsequent development work.

In chapter 3 the design considerations and theoretical foundations upon which the development is based are explored. This chapter describes how the system works in principle and proves that the solution is technologically feasible. Key areas of technical difficulty are identified and individually analysed.

Chapter 4 describes the hardware of the TreeScan system, with an emphasis on the custom designed scanner. Functional blocks of the scanner are described in detail and the reasons for this particular implementation are presented. In addition to this the obstacles encountered during hardware development are briefly discussed.

In chapter 5 the algorithms implemented in the TreeScan software are described. This includes both the image capture software used to capture images with the scanner and the parameter extraction software which is used to estimate actual tree dimensions.

Chapter 6 is an evaluation of the system accuracy and discusses the modifications made to convert the scanner, as originally designed and built, to an accurate scientific instrument.

In chapter 7 possible implications of this technology on the forestry industry are presented. Strengths and limitations of the TreeScan system are discussed and recommendations are made regarding the future directions for this research and development work.

To conclude the thesis, the main points of this research are summarised in chapter 8.

Relevant detailed technical documentation and software listings are included in the appendices.

Unless noted to the contrary in the text, all work is the authors own work. This includes; analysis of design considerations, system sensitivity analysis, design and testing of all digital hardware, design and testing of the majority of analog hardware, all microcontroller software development, and the majority of the system evaluation tasks.

The development of the image acquisition plug-in, the distortion correction methods, and the image processing macros were a joint effort between the author and his supervisor (Dr. Ralph Pugmire).

Notable tasks completed by other development team members were all mechanical engineering, design and testing of stepper motor controller, design and testing of power supply, and the final accuracy tests in two dimensions.

1.3 Forest Industry Background

The aim of this section is to provide a forestry background to set the context for this research. It is aimed at the reader with very little forestry experience, providing a brief overview of key aspects of the industry. It is intended to be an introduction and does not comprehensively cover all aspects of the forestry industry.

1.3.1 Introduction

Plantation forestry is the sector of forestry that deals with production forests. Production forests are forests specifically planted with the aim of being harvested. Plantation forestry does not include the felling of natural forests and is therefore a sustainable and renewable industry.

Plantation forestry is a major export industry of New Zealand. In 1993 the export of forestry products constituted New Zealand's third largest export earner, generating 2.5 billion dollars. This is almost on par with meat and dairy exports, 3.0 and 2.8 billion dollars respectively (Forestry Facts & Figures, 1994).

New Zealand production forests are predominantly radiata pine (90%), with smaller quantities of douglas fir, softwoods, and native hardwoods. The New Zealand radiata pine estate constitutes 34% of the global radiata pine estate (Forestry Facts & Figures, 1994). New Zealand radiata pine plays an important role in the New Zealand economy and constitutes a large proportion of the global radiata pine market.

The main plantation forestry area in New Zealand is the Rotorua district in the Central North Island with smaller scale forestry blocks scattered throughout the country. The ownership of these forests is divided between three large forestry companies and a significant number of smaller owners. The three largest owners are Fletcher Challenge Ltd., Carter Holt Harvey Ltd., and the Forestry Corporation of New Zealand. They own 16%, 25% and 13% of plantation forestry resources respectively (Forestry Facts & Figures, 1994).

The forestry industry has seen a phenomenal growth over the last three years. This is largely a result of increased international demand driving world timber prices up. As the value of sawn timber rises, the value of the raw product also rises and it becomes important to maximise the use of business resources. Good tree breakdown is no longer good enough, the tree breakdown must be optimal.

1.3.2 Forestry and Sawmilling

The timber industry traditionally contains a clear distinction of roles. The role of forestry operations is very different from the role of sawmilling operations.

- The role of **forestry operations** is to produce logs. In practise, forestry operations includes the planting, growing, and maintenance of the trees during the time they are growing. Once the trees are ready to harvest they are felled and cut into logs of one of a number of specified grades (see appendix C).
- The role of the **sawmilling operations** is to process logs. The sawmilling operations commence with the raw product of logs of a certain grade and process these into sawn timber and other wood products.

The result of this division in the industry is that a sub optimal resource optimisation may be achieved. If this division is reduced and the tree optimisation can be based on final timber usage rather than log breakdown, resource optimisation could be improved. Many companies are currently restructuring to reduce this division.

1.3.3 Forest Operations

The basic unit of measurement in the forestry industry is the stand. A stand is a block of trees of similar age, size and other characteristics. Each forest is subdivided into even aged stands of typically 20 to 40 hectares. Stands are harvested as a whole at a tree age of 25 to 30 years.

The life cycle or rotation of a stand of radiata pine begins when the trees are planted. It is split into three phases, with an inventory assessment made during each phase:

- **Early growth** during which pruning and thinning operations may be completed
- **Mid rotation** during which the trees are left to grow largely unattended
- **Harvest** during which the trees are felled

The **early growth** phase, 0 - 10 years, determines the quality of the trees in a stand. Trees are pruned in successive lifts up to a maximum of 6 or 8 m. The result of pruning is trees which grow straight and have large sections of clearwood. Clearwood is wood which does not contain any knots or defects outside a defect core.

Stands will undergo two thinnings to select the best trees and reduce the stocking to a level that will produce a maximum tree growth rate. The first thinning is at a tree age of 4 to 6 years, the second at 7 to 9 years.

During the **mid rotation** phase, 10 - 25 years, very little tree maintenance is required. Generally the only task completed is the mid rotation inventory assessment.

During the **harvest** phase, 25 - 30 years, the trees are felled to produce stems. These stems are then cut into logs based on the current cutting strategy.

Once the trees in a stand are felled, the stems are taken to a skid site. A skid site is a small area of the stand which has been cleared and where the stems are cut into logs. Typically there will be several skid sites per stand. There are two primary methods of stem removal; the skidder and the hauler. A skidder is a large wheeled vehicle which drags the stems to the skid site. The hauler is a cable based pulling system which must be used when the terrain is too steep for a skidder.

On the skid site the stems are cut into logs. This breakdown is intended to optimise the use of a tree, but is a compromise between maximising value and meeting orders. The log maker decides on the best breakdown for a particular stem based on the log maker's assessment of stem shape and features, and the current log requirements. The total value of the recovered logs depends on the performance of the log makers. Generally the performance of a log maker is very good, typically 95% of optimal. If the performance of a log maker drops below this level, this results in a very large loss in stem value.

Once the stem is cut into logs they are stacked until they can be trucked out of the forest.

1.3.4 Inventory Assessment

Assessing the value and potential yield of a stand of trees is one of the basic concerns of commercial forest growers. During each of the three phases in the stand life cycle an inventory assessment is made. This involves gathering information on a representative sample of trees from a particular stand.

The first inventory assessment is made during early growth phase, at a tree age of 4-10 years. This is the **quality control inventory** which allows the forest owner to check that the pruning and thinning have been completed properly. Basic information is collected regarding the condition of the stand as a whole such as total tree stocking, tree diameter, tree height and the pruned height.

At a tree age of 15-16 years the **mid rotation inventory** assessment is made. This enables the owner of the forest to gain information on the growth progress of the trees.

The **preharvest inventory** assessment is made 1-2 years prior to harvest, at a tree age of 23-28 years. The main aim of this final inventory assessment is to aid in market planning and harvest scheduling. Information is collected regarding the stocking of the stand as well as detailed information regarding the characteristics of individual trees. Section 1.4 will discuss the preharvest inventory assessment in greater detail.

1.4 Preharvest Inventory Assessment

As stands mature growers require detailed inventory information to plan harvesting, marketing and utilisation of the timber. Logs are cut on a 'to order' basis, with no buffering of stock on hand. This implies that good inventory information is necessary to determine what log grades can be expected from a particular stand. On a short term basis if there are not enough logs to meet a particular order, higher quality logs may be downgraded to fill the outstanding order. The result of this is a serious loss in profitability.

The aim of the preharvest inventory is to provide information regarding the value and quality of individual stands. This information is used in :

- **Harvest planning** - The log grades which can be most profitably cut from a stand are estimated. Harvesting operations are planned based on which stand can provide the optimal log grades to meet particular orders.
- **Market planning** - The volume of harvest, by log grade, is estimated up to three years ahead of harvest. Export contracts are based on the estimated volume of harvest.
- **Valuation** - The absolute value of a particular forest block can be estimated from the inventory information. The value of a forest may need to be established if the forest is sold or if company assets are valued.

The assessment of total volume and quality should be based on the actual measured condition of the trees. The effect of disease and damage, and management operations such as pruning and thinning must be directly taken into account. However the data collected should be flexible enough to allow harvest to be estimated even if log specifications change after the inventory team has visited the stand.

There are two important aspects of any tree which must be measured in order to be able to estimate the optimal log breakdown; the shape of the stem, and the quality of the stem. The shape of the stem, or sinuosity, is defined by the amount of sweep and wobble the tree has (see glossary or appendix C). The quality of the stem is defined by the branch sizes, pruned height and defects such as rot, broken tops, forks and nodal swelling.

Currently the 'MARVL' (Method of Assessment based on Recoverable Volume by Log type) system is being used by most major forestry companies in New Zealand. MARVL is an inventory assessment method designed specifically for the preharvest inventory.

1.4.1 MARVL Inventory Assessment

MARVL was developed by the Forestry Research Institute of New Zealand in the 1970's in response to the need for a general purpose inventory tool and is now widely used in Australia and the Pacific as well as New Zealand. It is based on the visual assessment of a sample of trees. In addition to the visual assessment, a number of tree parameters are measured. From this information log production estimates are calculated.

The MARVL system is a general purpose method which has been designed to allow flexibility in its use (Deadman & Goulding, 1979). As result, each user of the MARVL system has a slightly different implementation. The MARVL system involves three steps: sampling, cruising, and estimating log production.

Sampling

A series of bounded plots are defined as a representative sample of the stand. Each plot covers an area of 0.04 to 0.06 hectare, with a total of approximately 4% of the stand area sampled. The number of plots per stand is based on stand area. A typical number of plots per stand is 15, but can vary from 10 to 100.

Cruising

Once the plots have been established, a team of two people is sent out to assess each plot. During the assessment the heights of two trees are measured using a clinometer, stem diameter at breast height is measured for each tree and an visual assessment is made for each tree.

The visual assessment estimates sinuosity (in three classes of sweep), and quality features from the base of the tree. The sinuosity of the tree is recorded by describing the stem as consisting of sections of estimated length with a given sweep class and branch size class. For example:

- Sweep may be classified into three classes: $<SED/4$, $SED/4-SED/2$, $>SED/2$.
- Branch size may be classified into three classes: <7 cm, 7-14 cm, >14 cm.

There are a large number of quality features. Quality features include pruned height and other defects such as rot, broken tops and forks (see appendix C). The height of each feature of interest is estimated and recorded.

The measured and estimated parameters are entered into a portable computer used as a data logger during the work out in the field.

Estimating log production

The recorded data is down-loaded from the data logger onto a computer running the MARVL software to estimate log production. If necessary the trees are "grown on" to harvest age, using growth models. The data from each tree is individually processed to calculate the best log breakdown. Essential cuts such as those at the position of forks and stumps are made first with simulated felling breakage if required. The resulting yield for the plots is statistically extrapolated to provide an estimate of recoverable volume by log type of the entire stand.

1.4.2 Weaknesses of the MARVL System

The MARVL system provides essential information, however it has limitations. Several aspects of the assessment are subjective, and the system has been developed to the point where it is limited by this subjectivity. This has been the result of an increasing need for more detailed and accurate inventory information and a greater variety of markets since the system was developed.

The log volumes actually cut from a particular stand often do not match the log volumes as predicted by MARVL. The total volume estimate is very good, typically within $\pm 5\%$, but the breakdown of this the volume by individual log types may vary between $\pm 10\%$ to $\pm 80\%$. This is not solely due to the limitations of MARVL as logs actually cut depend on a large number of interrelated factors. For example, a sub optimal cutting strategy may have been used intentionally to fill a particular order.

The results of MARVL depend on:

- The ability of crews to accurately estimate tree parameters.
- Information loss inherent in the method of calling particular trees.
- The ability of the MARVL software to extract the desired information from the recorded descriptions.
- Size of the sample and how representative this is of the stand.

MARVL depends largely on the accuracy of **human estimates**. Branch class, tree height and the height of quality features can be surprisingly accurately identified. The greatest limitation of MARVL is that the estimation of sweep is subjective. Estimation of sweep is difficult as it involves making an estimate of sinuosity in two dimensions for sections of the tree. Two different people calling the same tree can give two different classifications.

A second limitation of the MARVL system is that there is **information loss** when describing a tree. Ambiguity can develop if all the relevant information is not retained.

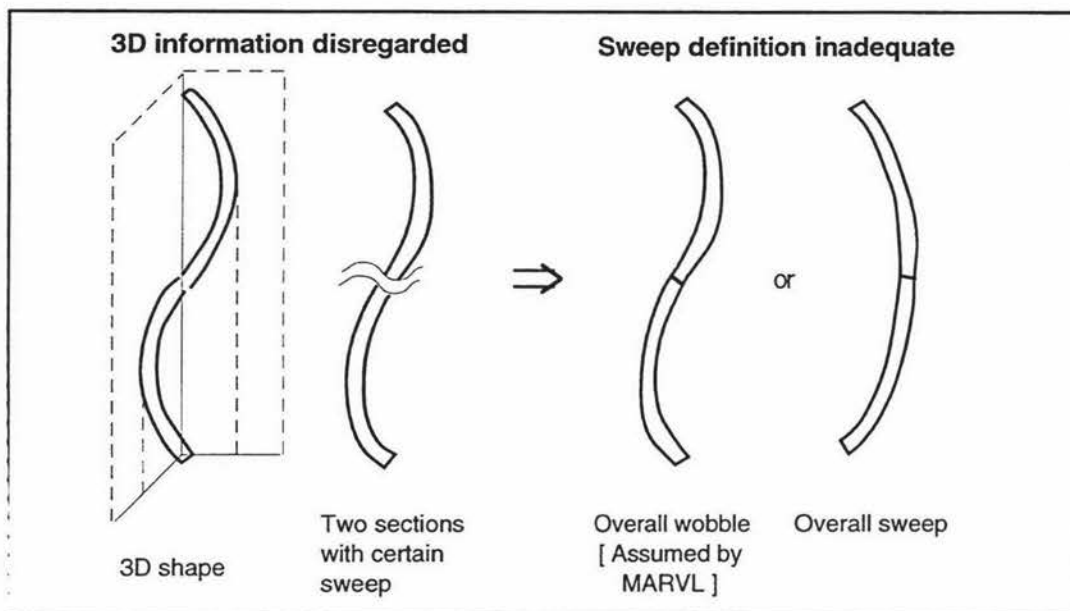


Figure 1.1 - Information loss inherent in the MARVL tree description

two examples of this ambiguity are:

1. MARVL is a two dimensional system. Three dimensional information is disregarded. If a tree is called as consisting of two sections of certain sweep, this could indicate the sweep is in the same plane or at right angles. This information which has an important impact on the optimisation of stem breakdown is disregarded (see figure 1.1).
2. Secondly the definition of sweep for different log lengths is inadequate. If a tree is called as consisting of two sections of certain sweep, this could indicate one of two situations. The tree could have a large sweep over the combined length or the tree could have wobble over the combined length (see figure 1.1).

In the above situation MARVL is not able to extract necessary information from the log description, so a simplifying assumption is made. MARVL assumes that a long log of the same sweep class as the greatest sweep of its subsections can be cut. i.e. that the log contains wobble in a single plane.

Lastly the sample must be statistically representative of the stand. Often there is a large variation of tree growth even within a stand. As a result the sampling procedure or stand area sampled may need to be modified.

1.4.3 Possible Improvements

In order to improve the forecasting system, one or more of the above weaknesses must be targeted for improvement. If a more quantitative system can be provided which does not suffer from loss of information, this would mean a great improvement for the preharvest inventory assessment.

A key consideration to maintaining high quality while retaining MARVL, is feedback to the staff involved regarding the results of subjective estimates. The more frequent and precise this feedback is the more successful it will be in maintaining the accuracy of the subjective assessment. However this is difficult in the assessment of sweep. The only reference to compare an assessment of a single tree, is how a more experienced person would call the tree. Even if the tree is felled, the extent to which the sweep was called correctly is difficult to determine.

In chapter two, several methods are proposed that can be used to improve the inventory accuracy using a partially automated system. This will produce more accurate and repeatable results.

Chapter 2

IMPROVED FOREST STAND ASSESSMENT DESIGN PROPOSAL

2.1	In-field Tree Imaging to Improve Stand Assessment	14
2.2	General Technology Options	16
2.3	Design Constraints	21
2.4	Proposed System	24
2.5	Design specification	26

This chapter reviews alternative imaging methods which could be used to improve inventory assessment. The approaches identified by the Massey University feasibility study are discussed in detail and analysed for design constraints. A final design proposal is put forward providing the basis for the rest of the development.

2.1 In-field Tree Imaging to Improve Stand Assessment

Over recent years the increase in computing power and improved digital image processing techniques have led to the exponential growth of electronic tools making use of digital image processing in many commercial and industrial applications.

The sawmilling operations of the timber industry have seen much development of scanning technology within the sawmill. This includes the scanning of log profile for optimised cross cutting, plank profiling for optimisation during edging and trimming, and internal scanning of logs to detect defect core structure. However this technology has made very little impact in the forestry industry. With in-field tree imaging this is changing.

In-field tree imaging is a method proposed to apply digital imaging technology to improve and automate forest stand assessment. This method involves generating a three dimensional model of a standing tree by capturing one or more images from ground level. This three dimensional model will provide more quantitative inventory information.

The three dimensional model fully defines the tree shape including sweep. This addresses the two main weaknesses of the MARVL based system as discussed in chapter one; the need to make a visual assessment of sweep in the field disappears and so human subjectivity is eliminated and there is no information loss. By removing the human subjectivity, the inventory system will be able to make more repeatable and accurate estimates of tree dimensions and as a result more quantitative inventory information will be available as basis for stand yield estimates.

Requirements of In-field Tree Imaging

If in-field tree imaging is to succeed the developers of a system will need to overcome a number of obstacles. These include the substantial technical difficulties in producing a working system that is robust enough to operate in a hostile forest environment.

The requirements of an in-field tree imaging system are that it must :

1. Work under forest conditions.
2. Produce accurate and repeatable results.
3. Be usable and productive.

A system must be capable of working in conditions typically experienced in a forest. This includes dense tree stocking, terrain and undergrowth variation, outdoor weather conditions, tree movement in the wind, and low levels of ambient lighting.

The system must be technically capable of producing accurate results that are repeatable regardless of expected variation in normal operating conditions.

The first two requirements are most important at this stage, but the system must also be usable and productive. This involves ease of operation by non technical users, acceptable portability (weight and ease of carrying), and acceptable productivity compared with existing methods of inventory.

2.1.1 Previous Research on Standing Tree Imaging

As the inventory of forests is a universal problem it was expected that there would have been previous research on the use of imaging techniques to improve inventory methods. This was not the case. After a comprehensive search no reference has been found to any other development work in tree imaging, tree sizing, tree assessment, or forest inventory assessment using imaging techniques.

The only reference found to other work specifically on standing tree imaging is a project by the Forest Research Institute of New Zealand, which is researching technology to solve the same problem as the development at Massey University. This was started approximately 18 months after development at Massey University was started.

2.2 General Technology Options

There are a large number of technology options that could be employed to develop a system to meet the above requirements. However, no technology can be said to provide the best or the worst solution. The use of each technology has advantages and disadvantages which must be considered. What is required is the lowest cost, easily developed, sufficiently accurate technology. This involves judgements by the developers based on past experience.

The most important consideration is a cost versus technology trade-off. Certain technologies may be faster or more accurate, however the cost in development expense and development time may be much higher. The requirements and benefits need to be carefully weighed up against the funds available to determine the best technologically feasible solution.

In the measurement of standing trees some form of remote sensing technique is required as the tree cannot be directly physically measured without climbing the tree or felling it, either of which would be unsatisfactory. The options available are some form of imaging using electromagnetic imaging (visual, infrared, Xray, or radar) or ultrasonic imaging. Each imaging method is briefly examined for technical feasibility in figure 2.1.

The imaging method employed must capture three dimensional tree information. Some imaging methods, such as laser and ultrasonic imaging, can directly incorporate three dimensional information. Other imaging methods are inherently two dimensional and the three dimensional information must be captured by some other means. Techniques that could be used are stereo imaging, multiple views from different directions, or structured lighting techniques.

The most promising solution is to use some form of imaging technique based on the visible spectrum. Investigation should be made into both the use of photographic systems and CCD based technology to capture images. The use of either infrared or ultraviolet imaging might also prove useful and should be investigated.

2.2.1 Massey University Feasibility Study

In late 1993 the Department of Production Technology at Massey University was commissioned to complete a study to determine the feasibility of using imaging techniques for the automation of forest stand assessment. In particular, to identify one or more approaches that could be developed to the prototype stage.

Photographic imaging - *System to capture visual images photographically*

Technically feasible and technology readily available, but working at the limit of normal resolution. Photographic film is only a temporary medium as the images require transfer to computer for later processing. Low technology cost but high per use cost with processing time delay. Need to capture several images at right angles to capture 3D information.

CCD imaging - *System to directly capture visual images electronically*

Technically feasible, but working past the limit of normal resolution, so may need to employ resourceful techniques. Medium technology development cost. Images captured directly into computer so low per image cost and no processing delay. Need to capture several images at an angle to get 3D information. Large quantities of data involved so a high powered computer or video tape required to capture and store large quantities of data in the field.

Laser imaging - *System to capture 3D locations of points on tree stem*

Probably technically feasible but a slow and very fragile system. High technology development cost. No images are required as 3D points are captured directly. 3D points can be captured directly into computer. Low powered computer required in the field with specialised imaging hardware.

Ultrasonic - *System to build up images by the reflected high frequency sound*

Can determine distance of objects in addition to direction. Technologically not feasible to achieve desired resolution. Moderate hardware development cost and low per image cost. High powered computer required to capture data in the field.

Xray imaging - *System to capture internal and external information*

Technically not feasible as the detector needs to be directly on the other side of the object being imaged. Very high development cost and image reconstruction techniques required to recreate tree information. Xray danger to the operator.

Radar imaging - *System to build up an image by reflected radar waves*

Can determine distance and direction. Technically not feasible as non metallic objects give poor radar echoes. Very difficult to get desired resolution.

Figure 2.1 - Imaging technologies

Investigation proved that no existing electronic imaging system existed that met the requirements of this particular application. No existing system had an appropriate aspect ratio or adequate resolution. After experimenting using a photographic method of capturing images, two approaches for capturing images directly into a computer were identified as favourable solutions that could be taken to the prototype stage (Pugmire, 1993).

1. The most promising approach used a **line scan camera** that was stepped through a series of fixed angles to build up an image of the tree. A single image would be captured that contained significant perspective distortion. The image would be recorded directly on to portable computer for immediate processing. This would achieve an image resolution of 500 by 8000 pixels.
2. The second approach used a **video camera** to capture a series of images while recording camera tilt. This method would use a domestic video camera and store the information on video tape for later processing. This system is likely to be less development intensive but greater image processing is required. Individual video frames must be spliced together to form a single image, and perspective distortions within and between frames must be corrected for.

The first approach is based on the premise of immediate processing by a system in the field, while the second approach is based on the premise of capturing lower quality images in the field and more extensive processing later. Both approaches were considered suitable to take into the prototype phase.

In addition, methods for extracting tree parameters from the images were investigated. Using an operator assisted method of parameter extraction, suitable methods for image calibration and perspective distortion correction were determined. Heights, widths, branch size and position estimates could be relatively easily calculated. The possibility of automated parameter extraction was deemed to require further research.

2.2.2 Alternative Approach

In addition to the approaches highlighted by the feasibility study a third image capture approach, based on a **high resolution CCD area camera**, must be considered as a likely solution.

The most important reason existing systems cannot be used for this application is that they do not have an aspect ratio of 40:1. The specifications require a degree of accuracy in the horizontal direction and a somewhat lower resolution in the vertical direction. The resolution requirement of 8000 by 1000 pixels is based on equal resolution in the horizontal and vertical direction. If a compression lens or a curved mirror is used the image can be compressed in the vertical direction without loss of resolution in the horizontal direction.

A high resolution CCD area sensor (for example 2000 by 1000) could be used instead of a line scan sensor and obtain similar results for sweep estimation as the system 8000 by 1000 line scan system. This system has the advantage that the image capture involves one integration period only. As a result the integration period of the image capture will be 8000 times faster (2000 x 4).

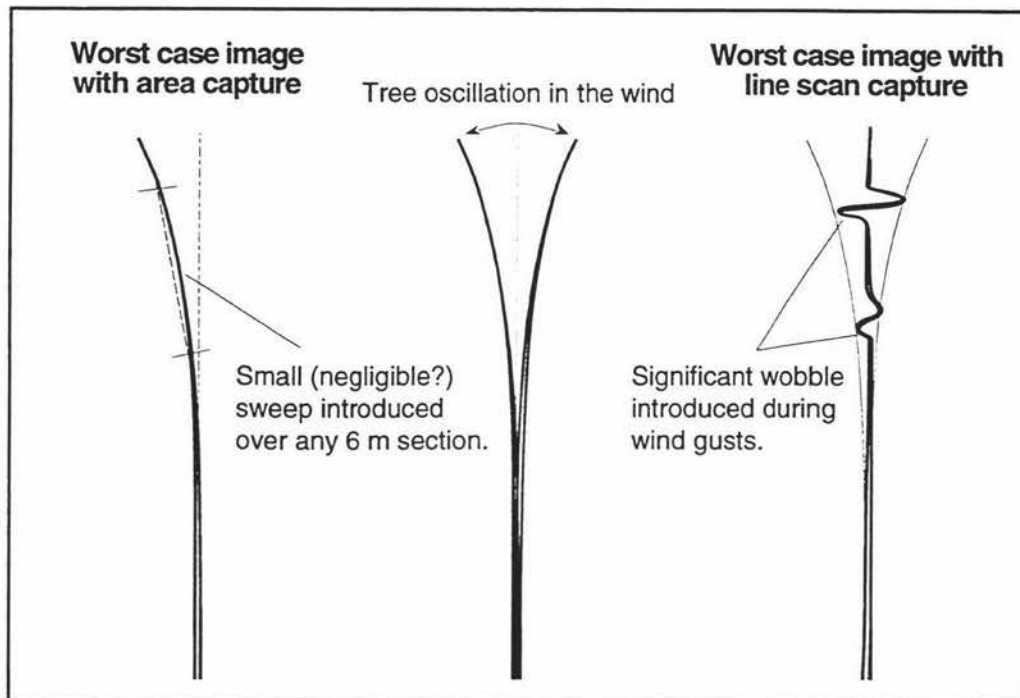


Figure 2.2 - Effects of wind on captured images

A second advantage of this system is that the shape of the tree stem would be instantly captured. If the tree was moving in the wind, given a worst case scenario, the image could be captured at the point of maximum tree deflection introducing a small amount of 'apparent sweep' (see figure 2.2). With the line scan approach however the image is built up over time, tree movement in the wind could introduce a large 'apparent wobble' that may be difficult to distinguish from real stem deformation.

Another advantage of this approach would be that the captured image is smaller. This would make it easier to process and store. One of the basic premises of image processing is reducing processing requirements by minimising raw data.

This approach would be limited in the measurement of branch sizes and may not provide adequate vertical resolution near the top of the tree. Use of an alternative system for the measurement of branches could be considered.

Figure 2.3 summarises the three alternative image capture approaches. It was decided that a prototype system based on the line scan approach should be built as this was the highest resolution system and provided the greatest flexibility for control of the system parameters during image capture.

The next section will investigate the constraints that are imposed on such a system and that will need to be considered for the developed system to be successful.

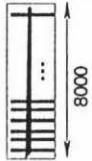
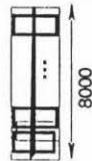
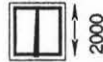
Property	Line scan approach	Video scan approach	High res. CCD approach
			
Scan time at 10 mS per exposure	80 seconds	20 seconds	0.01 seconds + 1 second transfer
Image size - Pixels	8000 x 1000	8000 x 500	2000 x 1000
- Storage	8 megabytes	4 megabytes	2 megabytes
Storage media	Computer hard disk	Video tape	Computer hard disk
Depth of field	Small	Average	Large
Aperture	Wide open	Average	Small
Development - Time	Medium	Low	High
- Cost	Medium	Low	High
Measure - Sweep	Yes	Yes	Yes
- Branch size	Yes (horizontal / vertical)	Yes (horizontal / vertical)	No (horizontal only)
- Height	Yes	Yes	Yes
Advantages	Can adjust aperture during scan Can adjust focus during scan	Fast image capture No computer req'd in field	Instantaneous capture No mechanical moving parts
Disadvantages	Slow as 8000 images captured Tree may move in wind	Image splicing / processing req'd Tree may move in wind	Difficult to develop Reduced vertical resolution

Figure 2.3 - Alternative image capture approaches

2.3 Design Constraints

Now that three specific solutions have been proposed, it is important to investigate the constraints imposed on a system. These constraints fall in two broad categories; constraints that result from the **forest work environment**, and constraints that are a result of fundamental **technology limitations** (see table 2.1).

It is very important that all constraints are considered during the design of a system, as any one of the constraints is able to reduce the usefulness of the final system. Each constraint will be discussed in detail with examples drawn from the image capture systems proposed in section 2.2.

2.3.1 Constraints Imposed by Forest Conditions

The system must be capable of operating in normal forest conditions. This imposes seven constraints that need to be considered. Each constraint is discussed below:

System robustness - Any implemented system must be rugged and able to withstand the knocks and vibration of work in the forestry industry. The intended users are accustomed to handling heavy duty forestry equipment and may not be accustomed to the sensitivity of electronic equipment. The system designer must take this into consideration.

Tree stocking - Tree stocking varies from 200 to 800 stems per hectare. The stocking of the stand limits the image capture positions that can be used. If the imaging system is too close to the tree, visibility of tree trunk near the top will be obscured by the tree's own branches. If the imaging system is too far away the top of the tree trunk will be

<i>System Constraints</i>	
Forest conditions	Technology limitations
System robustness	Resolution
Tree stocking	Aspect ratio
Terrain and undergrowth	Perspective distortion
Outdoor weather conditions	Image size
Tree wind movement	Image storage requirements
Lighting and contrast levels	Tree parameter extraction

Table 2.1 - System design constraints

obscured by the branches of other trees. A compromise needs to be made where the most important section of the tree stem can be reliably imaged without getting obscured.

Terrain and undergrowth - Terrain that must be inventoried varies from steep (slope of 40 degrees) to flat and there is normally some undergrowth present. Currently any significant undergrowth is cleared within the plot before the inventory assessment is made. A system should be capable of being used in situations where the terrain is steep and the undergrowth is present.

Outdoor weather conditions - A system must be reasonably weatherproof for two reasons. Current inventory crews work all year around, so a lot of the time it will be raining or misty. Secondly the undergrowth in the forest floor will usually be wet for a large part of the day. A system should be able to withstand a reasonable degree of moisture.

Tree wind movement - Trees do not remain perfectly still to allow image capture over a long time frame. For example with a wind of approximately ten knots the tops of the trees in an exposed stand will move up to one metre with an oscillation period of around five seconds. This is significant as the stem position is being imaged to within ± 1 cm.

Lighting and contrast levels - Overall lighting conditions in a forest vary greatly. There are changes in ambient lighting from very dark to very light depending on weather conditions and density of foliage. In bright daylight the conditions are good for normal photography. In overcast conditions the lighting is barely adequate for normal photography. There is also a large variation in contrast and lighting between the top and bottom of a typical tree. Near the bottom the image will be of a low light, low contrast, front lit object. Near the top of the tree the image will be of a high contrast, high lighting, back lit situation. It is difficult to capture good images in these conditions with any imaging system.

2.3.2 Constraints Imposed by Technology Limitations

Design constraints imposed by fundamental technology limitations must also be considered. The system must be based on practical technology and physically capable of delivering accurate results regardless of expected variation in operating conditions. There are seven technology constraints :

Resolution - An accuracy requirement of ± 1 cm near the top of the tree requires a minimal resolution of about 0.5 cm per pixel. This relates to an image resolution of approximately 8000 pixels by 1000 pixels. At the bottom the pixel resolution will be 0.2 cm per pixel. This is not available in any existing imaging systems. Normal CCD video cameras use an image resolution of approximately 500 by 400, with digital

camera technology typically using a resolution of 2000 by 1500 pixels. Line scan CCD cameras are available from 128 pixels to 8000 pixels.

Aspect ratio - The aspect ratio of a standing tree is approximately 80:1. As a result of the perspective distortion a desirable image aspect ratio is approximately 8:1. Images can be captured at this aspect ratio or techniques could be used to capture images of a more standard aspect ratio. This could be achieved through the use of a non circular lens such as those used by the wide screen cinema industry or through the use of a curved mirror as discussed in the high resolution CCD area camera approach.

Perspective distortion - The tree being imaged is viewed from below introducing a perspective distortion that can be corrected ^{by} for using a calculation, if the geometry of the imaging situation is accurately known. This may be achieved through the use of a calibration object in the captured images. Distortion correction is more difficult for the video approach than the line scan approach.

Image size - The sheer size of the tree images of these dimension make the images very difficult to work with. If an image consists of 8000 by 1000 pixels at 8 bit greyscale this corresponds to 8 megabytes of data. With images this size the computing power required to load, save and process the images is large. This is important as processing power of computers is limited. With the video scan system, overlapping images must first be spliced together and then processed. This will make the processing requirements even higher, but can be completed using batch processing out of the forest.

Image storage requirements - The data storage requirements for images of this size are very high. A typical high performance portable computer may contain 160 megabytes of hard disk space. This is the equivalent of 20 images or ten imaged trees. In contrast MARVL information for the same tree consists of approximately 100 bytes of text. This is one 80 000th the size of a single image. Video tape as used by the video scan approach is a very cost effective method of storing large quantities of image data.

Tree parameter extraction - The captured image is only raw data. Tree stem size information must be extracted from the image so a three dimensional model can be generated. This involves some form of processing of the raw data (image) to get out the desired tree size information. This is a task humans can complete readily but is difficult to automate.

2.4 Proposed System

2.4.1 Improved Assessment Outline

The image capture system is the first part of a sequence of steps in the proposed method for improved forest stand assessment. The images captured need to be processed to extract the tree parameters which define the dimensions of the three dimensional model. This model can then be processed by an optimiser to determine the optimal log breakdown for a particular cutting strategy. The recoverable yield and value of the stand for the optimal log breakdown can then be predicted (see figure 2.4).

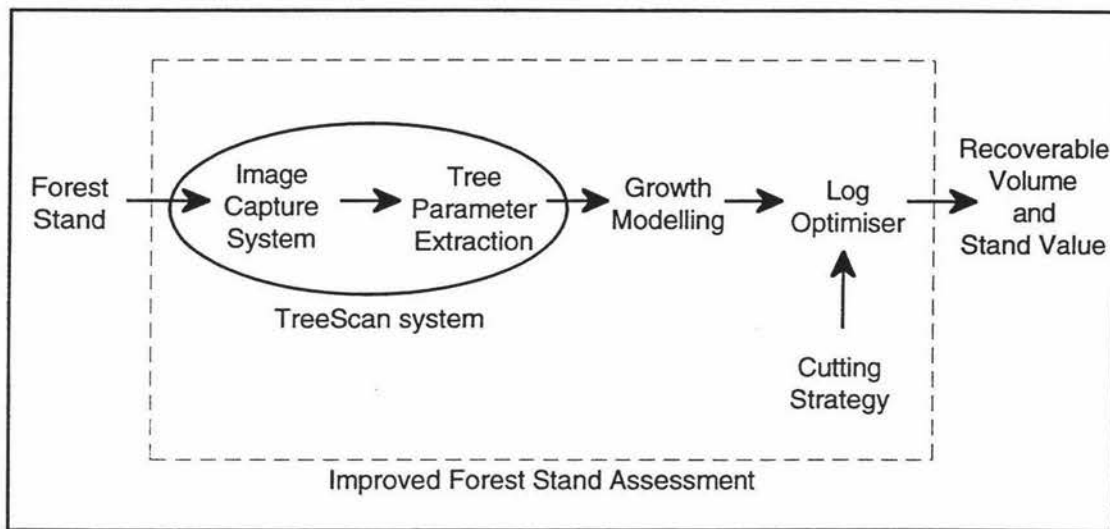


Figure 2.4 - Improved forest stand assessment overview

The proposed TreeScan imaging system works by capturing calibrated images of a tree. Based on calibration data and the position of features in the image, the position of the tree features and shape of the stem can be estimated. This is the tree parameter extraction. By capturing two images at right angles a three dimensional model of each tree is generated.

2.4.2 Proposed Image Capture System Overview

The proposed image capture system is a custom designed scanner capable of capturing high resolution images with a high aspect ratio directly into a computer.

The scanner would use a line scan camera that is stepped through a series of fixed angles to build up an image of the tree (as outlined in section 7.3 of the feasibility study - Pugmire, 1993). The system would capture a single horizontal scan line for each position of the rotating mechanism. The consequences of this are that the image is slowly built up one line at a time as the mechanism rotates (see figure 2.5).

Either a rotating lens and camera unit, or a fixed camera with a rotating mirror or prism could be used. A rotating mirror has the advantage that the sensor and cabling can be fixed and that the mechanism has less mass to rotate.

The image would be recorded directly into a portable computer eliminating the need for storage on photographic film or other temporary medium. A portable computer would be taken into the field during image capture, so the image could be immediately processed if required.

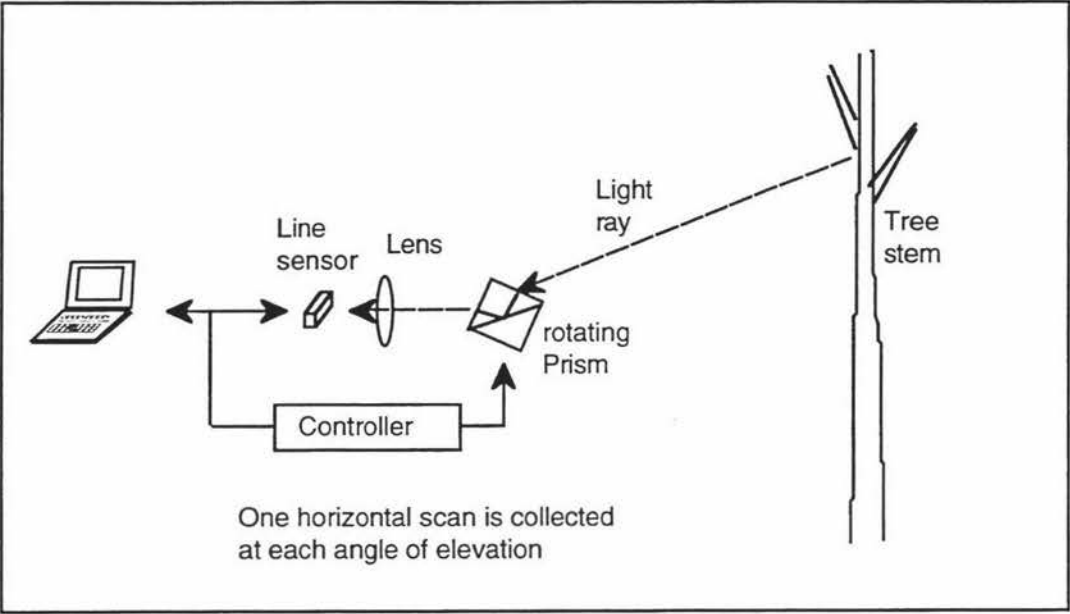


Figure 2.5 - Image capture system principle

2.5 Design specification

The aim of current work by Massey University has been to develop a prototype line scan based image capture system for the improvement and automation of forest stand assessment as specified in "Line Scan Camera Image Capture Project - Sub Project 1 Proposal" (see appendix D).

It was the intention of the project to rapidly produce a working proof of concept prototype to allow the capture of images of trees and transfer these directly to a portable computer. The captured images would be in a format that could initially be analysed using the NIH Image package and the macros produced as part of the feasibility study.

The system needed to be capable of imaging ten trees per hour under normal forest conditions. Normal forest conditions include low natural lighting, variation in tree dimensions and in tree stocking, presence of undergrowth, and a large variation in terrain.

The accuracy specifications state that the height estimates need to be accurate to within ± 0.5 metres and stem diameter estimates need to be accurate to within ± 1 cm. This relates to an image resolution of approximately 8000 by 200 for a 40 metre tree.

Imaging the bottom two thirds of the stem may be sufficient as this is the most valuable section. The top of the tree would normally be obscured by branches. In addition to the stem sweep determination, estimation of branch size was classified as being desirable.

The initial aim was to develop a prototype system by July 1994. Due to several development obstacles the prototype system was delayed until November 1994.

Chapter 3

TREESCAN DESIGN CONSIDERATIONS AND THEORETICAL FOUNDATIONS

3.1	Design Overview -----	28
3.2	TreeScan Operating Principle -----	31
3.3	Image Capture -----	35
3.4	Scanner Interface and Image Storage -----	46
3.5	Parameter Extraction -----	48
3.6	Three Dimensional Model Generation -----	57
3.7	Implications of Image Capture Geometry -----	58

TreeScan is the name given to the prototype tree imaging system developed. The aim of this chapter is to show that the TreeScan design is a technologically feasible solution. This will be accomplished by discussing the design considerations and theoretical foundations of likely areas of technical difficulty. Key areas of technical difficulty are individually analysed. As a result each section in this chapter is an almost standalone analysis and discussion.

To provide an introduction section 3.1 and section 3.2 will discuss the TreeScan design at a systems level and show how individual design aspects interrelate, after which sections 3.3 to 3.7 will analyse individual design aspects in greater detail.

3.1 Design Overview

Once the direction of this research and development had been established the design phase was entered and research was started to determine the best method to realise the design concept. This research involved determining the limiting factors of the technology, calculating precise technology requirements, and scoping technologies currently on the market.

The TreeScan system will fit into the improvement of forest stand assessment as shown in figure 2.4 in the previous chapter. The system input boundary lies at the physical geometry of the trees that need to be measured. The system output boundary lies at the actual three dimensional tree model produced by the software. To get from input to the system output involves a series of steps (see figure 3.1) each one of which could affect the integrity of the information produced by the system and the feasibility of the whole design. Each of these steps is a possible area of technical difficulty which may limit the system and must be carefully analysed.

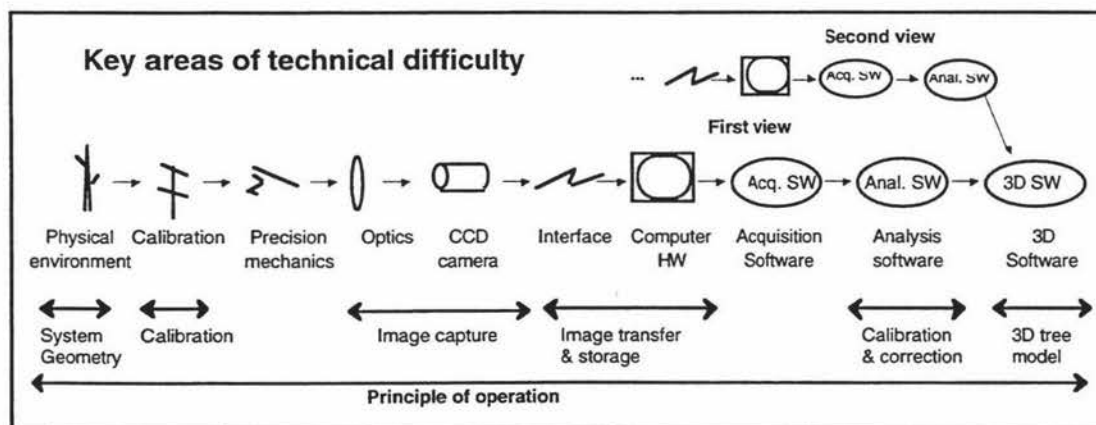


Figure 3.1 - Possible areas of technical difficulty

The principle of operation of the TreeScan system is discussed in section 3.2. This describes how the system works from a conceptual viewpoint. The discussion remains at a systems level and covers the TreeScan system from input to output.

The largest part of the design involved the design of the actual image capture system. In section 3.3 image capture technology is reviewed. This covers both the system optics and CCD technology. The differences between conventional imaging techniques (such as photographic and area CCD) and the TreeScan system are also discussed.

As the images that will be captured are very large, image transfer and storage requirements are an important consideration. These are discussed in section 3.4. Image transfer and storage requirements also have an impact on the computer that will be used.

Image calibration and the mathematical correction of various forms of image distortion are discussed in section 3.5. This includes the placing of a calibration rod of known dimensions in the image to determine the image capture geometry.

Section 3.6 discusses the generation of a three dimensional model from multiple views, and lastly the expected effects of the geometry of the imaging situation are calculated. This provides expected errors that the results of experiments with the TreeScan system can be compared against.

Section 3.11 provides a brief note on the systems integration aspects of the project.

3.1.1 Systems Integration Project

The design and development of a custom instrument such as the TreeScan scanner is primarily a systems integration task. Only by combining knowledge, theory, and hardware from a large number of specialist engineering disciplines is it possible to develop a successful system.

The main engineering disciplines that have been called on during design and development of TreeScan are:

- **Image processing** - For the use of image capture technology (CCD cameras) and image processing techniques.
- **Optics** - To determine lens and mirror requirements, and techniques for calculation image depth of field and resolution.
- **Mechanical engineering** - To machine the precision mechanisms required to rotate the mirror.
- **Electronics and computer interfacing** - To interface all individual components and provide control over the scanner functions.

- **Computer programming and software development** - Techniques for the custom computer software development and microcontroller programming.
- **Mathematics and photogrammetry** - To allow perspective distortions to be corrected for in software.
- **Product development** - For the overall design and usability of the system to meet the needs of the user.

In a project that draws from each of these engineering disciplines there is generally a team of people involved, as has been the case with the development of TreeScan. In such a team environment communication and project management tasks become as important, if not more so, than the technical aspects. Work being completed in each subsection must remain coordinated and team members must remain in constant communication with the rest of the project team to prevent misinterpretation or misunderstanding.

3.2 TreeScan Operating Principle

The TreeScan system is a system for the estimation of tree shape and dimensions. The TreeScan system is based on capturing calibrated images of a tree. A calibrated image is an image captured in a situation of known geometry (or in which the geometry can be derived from the image) that can be used to make estimates of real world object dimensions.

It is important to distinguish between the use of the terms **measurements** and **estimates**. The use of the term 'measurement' will imply the dimension has been physically measured, while the use of the term 'estimate' will refer to a dimension based on a calculation performed on other measurements. This implies that the image is measured and the calibration rod is measured. Tree dimensions are estimates calculated based on image measurements, some simplifying assumptions, and known calibration rod dimensions.

The calibration rod is an object of known dimensions that is used to determine the image capture geometry. Two calibration rods have been used; the first is a pole with two crossbars, the second is a pole with one crossbar and a reference circle. The discussion that follows holds for both calibration methods.

The plane through the calibration rod is the calibration plane. The calibration rod is placed against the tree so that the calibration plane lies as close as possible to the plane of the tree. The principal scanner axis must be perpendicular to the bottom crossbar of the calibration rod. The scanner should be positioned at a distance so that the majority

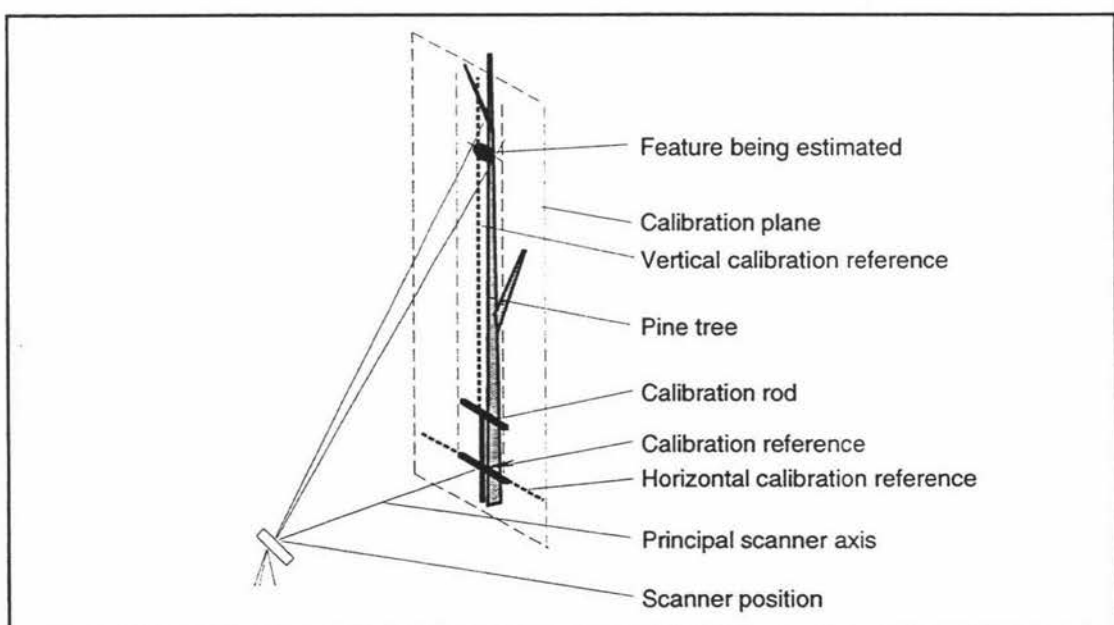


Figure 3.2 - TreeScan image capture

of the stem is visible. The range of 12 to 20 metres out from the tree provides a good position in typical tree stockings. The scanner position may be at any height with respect to the calibration rod.

The image captured contains perspective distortion, which implies tree feature dimensions cannot be directly measured from the image. A calculation, discussed in detail in section 3.5, is used to correct for this distortion and follows the principle that:

1. The **scanner position is estimated** by taking measurements from the image of the calibration rod, and making use of known dimensions of the calibration rod.
2. **Tree feature dimension estimates** are calculated using the estimated camera position and measured tree image dimensions.

The procedures being used are inherently two dimensional. When estimating the size of three dimensional objects, it is the size of the object's projection on to a two dimensional plane that is being estimated. If the three dimensional object does not lie exactly on the calibration plane an expected error will be introduced into the size estimates (see figure 3.3). This is discussed in greater detail in section 3.7. The use of Inclinometers for estimation tree heights suffers from the same limitation, which is inherent in the geometry.

Each image can provide information on the vertical axis (height) and one horizontal axis. By capturing multiple images at a known angle to each other, the system can be extended to deduce the three dimensional shape of objects. Two or more images from different directions can be combined to build up a three dimensional model of the tree. A minimum of two views of the tree are required, with improvements in accuracy as more

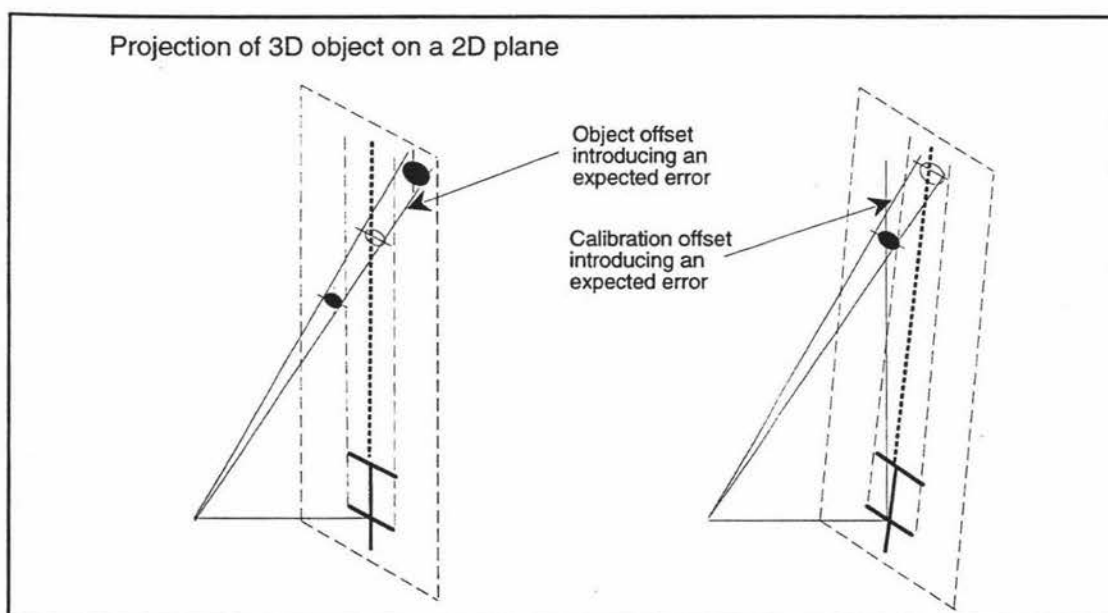


Figure 3.3 - Projection on a two dimensional plane

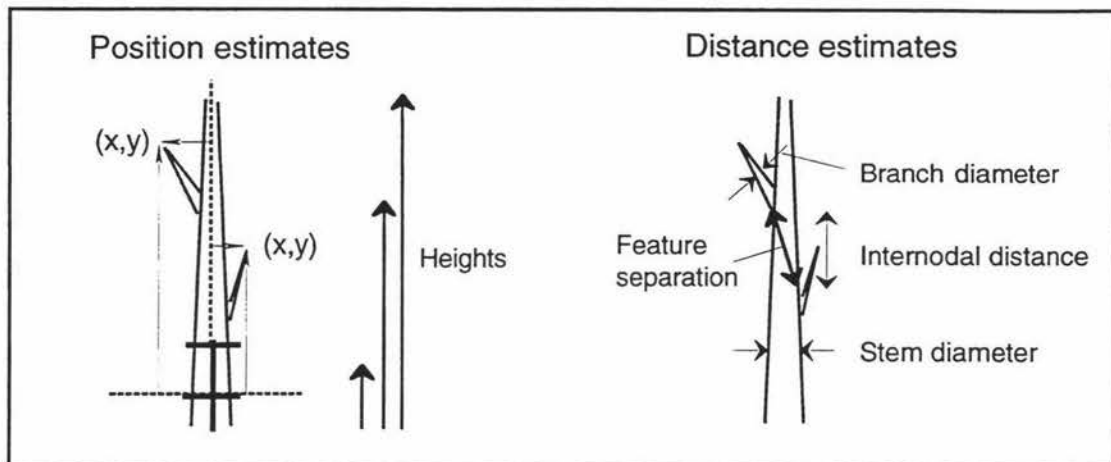


Figure 3.4 - TreeScan estimates

views are incorporated. If two images are used to build up the full model of the tree, the optimal view points are at right angles to each other.

3.2.1 TreeScan Estimates

The TreeScan system inherently makes **position estimates**. A position estimate is a point estimate that consists of a vertical offset and a horizontal offset from the calibration reference. Thus for any point of the tree a height from the calibration rod and a horizontal offset from the vertical calibration reference can be calculated. **Distance estimates** between two points on the tree can be calculated by taking the difference between two position estimates. All tree dimension estimates other than offsets from the calibration reference are distance estimates. Distance estimates can be either horizontal, vertical, or a combination of the two.

The TreeScan system is intended to estimate five tree parameters; height, sweep, stem diameter, branch diameter, and feature separation such as internodal distance.

- **Height estimates** are based on the vertical component of a position estimate and are calculated from ground level using the calibration reference.
- **Stem diameter, branch diameter, and feature separation** are distance estimates. Distance estimates can be made at any orientation.
- **Sweep** estimates are slightly more complicated. By definition (see appendix B) sweep is the amount the tree stem is offset from a straight line over a given length. This means that sweep is a combination between horizontal and vertical distance estimates.

It is important to consider what kind of estimates are being made because each is affected in a different way by the geometric inaccuracies as discussed in 3.7.

The model of the tree is defined by three dimensional stem midpoint position estimates and stem diameter estimates at fixed height intervals along the stem. These stem estimates are defined using the stem edges visible in each view. Stem diameter and stem midpoint are calculated using the difference in edge position and the average of the two edge positions respectively.

It should be noted that this method of describing the tree stem contrasts with the method currently used by the MARVL system which describes the tree in sections of **variable length with predefined sweep classes**. TreeScan does not define sweep classes, but implicitly defines sweep by providing **stem position information at fixed height intervals**. This reduces the loss of three dimensional information.

3.3 Image Capture

This section serves two purposes. First a background to digital image capture and CCD technology is provided for the forestry reader who may not be familiar with the operation of a digital image capture system, and secondly important image capture aspects of the TreeScan system are discussed. This includes important optical considerations as well as the difference between images captured using an area camera and images captured using a line scan approach such as the TreeScan system.

3.3.1 Digital Image Capture

Digital image capture is the conversion of light from a scene into an array of numbers inside a computer (a digital image), which consists of individual pixels (square blocks when zoomed right in). Each pixel has intensity value associated with it. By displaying pixels of different intensities in a rectangular grid the digital image can be viewed.

The conversion from light to a digital image requires a sequence of steps (see figure 3.5). Light from the scene is focused on a sensor (taking the place of the camera film), which converts the light into an analog voltage signal. The sensor contains many individual sensing elements (up to 1 million per square cm), each capturing the light for a single pixel. The sensor converts the continuous light signal into a voltage signal consisting of individual pulses (spatial quantisation). The number of elements on the sensor determines the resolution of the digital image.

The second step in digital image capture is the conversion of the analog voltage signals into a digital representation using an analog to digital converter. The analog to digital converter samples the intensity of the analog voltage of each pixel and converts it into a

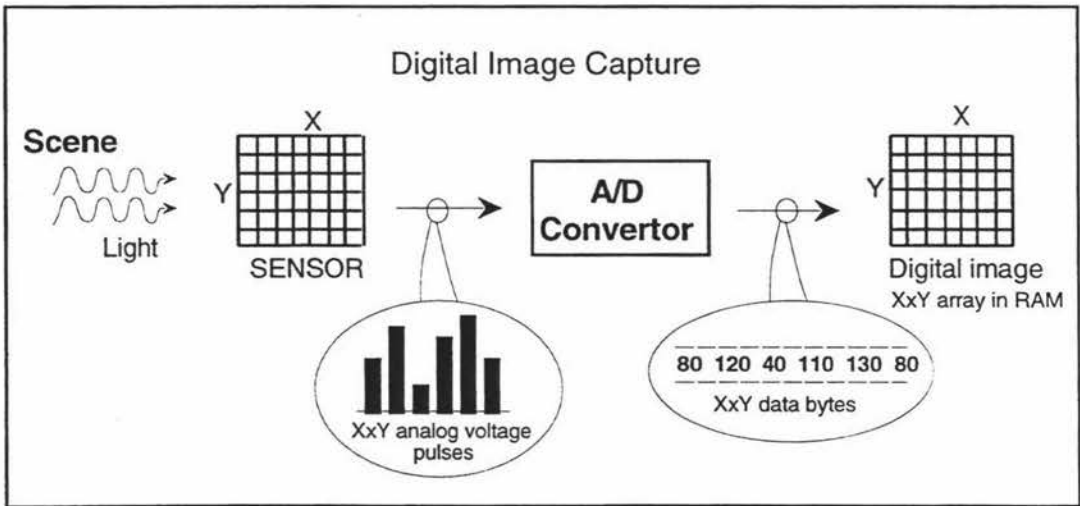


Figure 3.5 - Digital image capture

number. There are only a certain number of intensity values each pixel in the digital image can have (amplitude quantisation), which is typically an 8 bit number allowing 256 shades of grey to be discriminated. The numbers representing this digital image are then transferred to the computer and stored in an array in RAM (computer memory).

There are several types of sensor technology available including: CCD (Charge Coupled Device), CID (Charge Injection Device), and older vacuum tube sensors. The CCD sensor is the most commonly used including every domestic video camera.

3.3.1.1 CCD Technology

CCD sensors are electronic light integrating devices that generate a charge proportional to the exposed light intensity. CCD sensors are available in both linear one dimensional arrays and two dimensional area detector arrays.

CCD technology is based on the principle that photogenerated charge accumulated in a 'well' (defined by voltage potentials at the surface of a MOS structure) may be moved about by moving the local potential minimum. Electrons are accumulated under the transparent photogate, then transferred to a shift register so the data can be read out as a serial data stream.

The sensor consists of a P-type substrate of polycrystalline silicon with areas of N-type material at the surface (see figure 3.6). Over the semiconductor are a series of metal electrodes insulated by a layer of silicon dioxide. If one of the electrodes is energised this creates a depletion region or potential 'well'. Light passing through the transparent photogate electrode generates electrons. While the photogate is energised (during integration) these electrons collect in the depletion region under the photogate. As soon as the opaque transfer register electrode is energised collected electrons are transferred into a shift register which presents the data to the on-chip amplifier as a serial data stream.

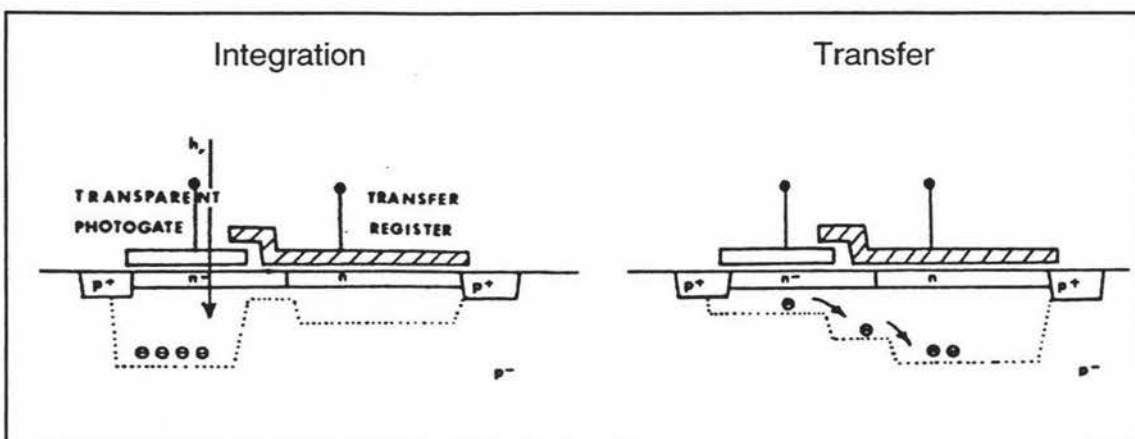


Figure 3.6 - CCD technology

3.3.2 Primary Imaging Considerations

When designing an image capture system there are a large number of considerations that need to be taken into account, but there are two fundamental principles that should govern the design of the system:

1. **Capture 'good' images** - The image capture conditions should be modified to capture images that are of the best possible quality for their intended purpose. It is relatively straight forward to ensure the captured image contains the desired information. It is more difficult to extract this information through the extensive processing of a poor quality image.
2. **Keep the image data content low** - Image processing is a computationally intensive process, lessened if the desired information can be extracted while keeping the raw image data content low.

With these general principles there are a number of factors that need to be considered when capturing images. The most important factors are:

Object illumination and contrast - This has a critical impact on the image captured. Changes in both illumination and contrast can improve image quality. In a forestry situation the lighting conditions are difficult to cope with, and difficult to modify. The use of a green filter is one possibility for enhancing the contrast near the bottom of the tree.

Resolution and accuracy - The image capture system must have the provision for sufficient resolution and accuracy to measure the desired image features. In a forestry situation branch size estimation will require a different accuracy than feature height estimation.

Avoidance of distortion - It is important to consider the possible distortion effects that may affect the system. Distortion can be introduced in several ways including both perspective distortion and any form of distortion introduced by the optical components.

Image calibration - Some method must be set up to enable measurements defined on the image to be translated into real world measurements.

Speed of image capture - The speed of image capture must be suited to the application. Ideally instantaneous image capture is desirable, but a trade off may need to be made against other factors such as cost.

3.3.3 Area Sensor vs. Line Scan Build-up

A system with an area camera such as a conventional photographic camera or an area CCD camera directly captures an entire image. The TreeScan scanner uses a tilting mirror mechanism to build up an image. This introduces a fundamental difference in the image generated. This difference is visually difficult to distinguish, but must be taken into account when taking size estimates from the image. The difference between the images captured with an area camera and the TreeScan scanner are discussed below.

3.3.3.1 Area camera

In an conventional photographic camera (also CCD area camera) the lens focuses an image of the object on to the film. More precisely the object is said to lie in the object plane, and the area in which the image is in sharp focus is called the image plane.

If the image plane is parallel to the object plane, the object is simply scaled down by the magnification factor to produce the image (see figure 3.7). Equal steps in the image plane relate to equal steps in the object plane and parallel lines on the object plane remain parallel in the image plane. This has the implication that a one meter object will cover the same number of pixels whether it lies near the top, or bottom, of the tree. Specialised photographic equipment is required to photograph tall objects like pine trees in this way.

Using normal photographic equipment, the image plane will not be parallel to the object plane, this introduces a perspective distortion. The perspective distortion is a linear distortion such that parallel lines in the object plane appear as straight lines, converging to a point at infinity in the image plane. This distortion must be corrected for when measuring objects from the image.

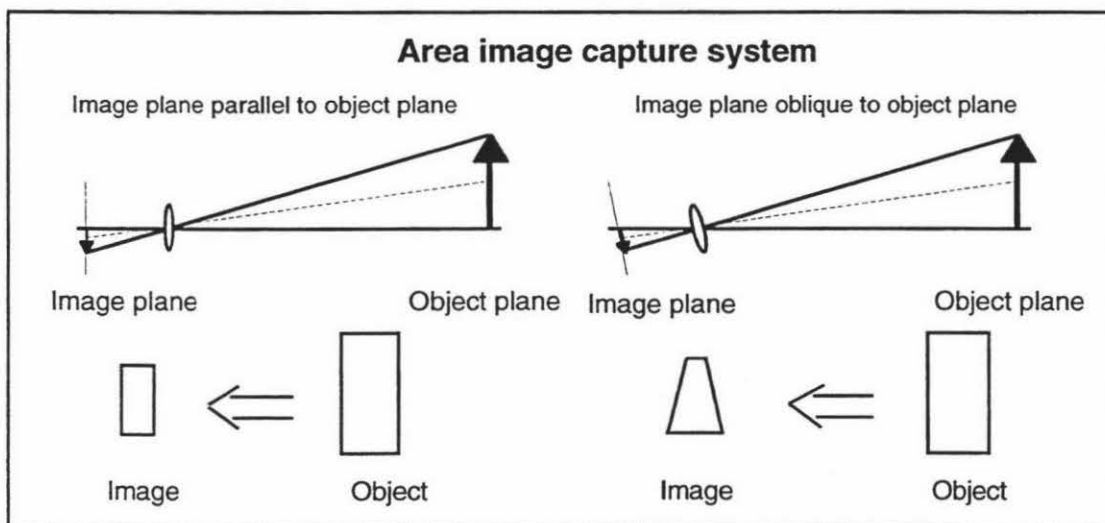


Figure 3.7 - Photographic image capture distortion

A photographic system has several limitations:

1. In order to capture a whole pine tree using normal photographic equipment from a distance of about 15 metres from the base of the tree a 28 mm wide angle lens is required. This introduces nonlinear lens distortions (see section 3.3.3) which must be compensated for when making size estimates from the image.
2. Any integration based system has the trade-off between depth of field and exposure time (see section 3.3.3). If the entire tree is imaged, a large depth of field (closed aperture) is required and a long exposure time must be used.
3. There is a large variation in lighting and contrast between the forest floor and canopy. In an area camera the same exposure must be used for the entire image.

3.3.3.2 TreeScan scanner

The TreeScan scanner uses a different approach to capturing images. The image is 'built up' one line at a time as a mirror is incrementally rotated. This introduces an additional distortion.

In the photographic system each image pixel is related to equal step sizes in the object plane. In the TreeScan system each image pixel represents a constant **angular step size** (see figure 3.8). The consequences of this are that pixels near the top of the tree will represent larger distances on the object. What visually appears to happen is that the top of the image tends to get squashed together. This is a nonlinear distortion which must be corrected for in the distortion correction software (see section 3.5).

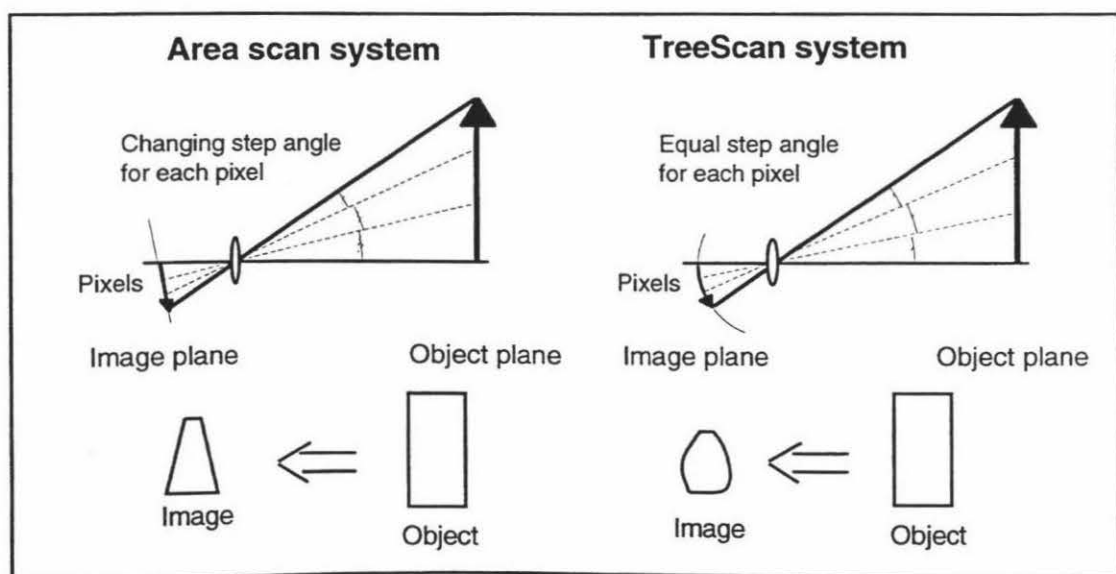


Figure 3.8 - TreeScan image capture distortion

A line scan approach such as the TreeScan system has several advantages over an area capture approach. These are that:

1. A lens with longer focal length can be used as the field of view required is smaller. In practise this means there will be no significant lens distortion.
2. The aperture and / or integration time can be adjusted during the scan to compensate for changing light and contrast levels.
3. The lens can be refocused during the scan up the tree so a large depth of field is not required. This allows for the use of larger apertures and hence shorter integration times.

The disadvantages of the line scan approach are that:

1. The image is built up using multiple exposures over time. The image capture will be slower than a single image capture and if the tree sways in the wind this may be indistinguishable from local shape deformations in the tree.
2. Distortions can be introduced owing to the misalignment of scanner components such as camera, mirror, and axis of rotation, and imprecision in the driving of the tilting mirror mechanism.

3.3.4 Optical Considerations

Image capture systems require a lens to focus an optical image on the image capture sensor. There are a number of factors that must be considered when deciding on a lens to use for a particular image capture application. There is the trade-off between aperture (hence depth of field) and exposure time as well as the considerations of lens quality. Lens quality is determined by the lens aberrations, lens modulation transfer function, and the lens' relative illumination.

A lens consists of one or more pieces of glass all of whose centres lie on a common axis (Horder, 1971). A lens consisting of a single piece of glass is a simple lens, and one consisting of multiple pieces of glass is a compound lens. Most practical camera lenses are compound lenses with typically three to seven elements. Lens principles can be visualised using a simple lens.

Light passing through a lens is limited by an aperture stop to control the exposure. The diameter of the aperture stop can be adjusted. The light passing ability of the lens is referred to as the relative aperture or f-number (Ray, 1979). Relative aperture is commonly referred to as 'aperture' or 'f-stop'. For a thin lens the relative aperture is the diameter of the aperture stop divided by the focal length of the lens. The aperture controls the brightness of the image on the film plane. Doubling the area of the aperture stop is referred to as one f-stop and doubles the amount of light coming into the camera. There is a standard series of f-numbers shown in table 3.1 for a 75 mm lens:

Aperture(N) = \frac{focal\ length(f)}{stop\ diameter(d)}

f-number	1	1.4	2	2.8	4	5.6	8	11	16	22
Aperture diam. (mm)	76	54	38	27	19	13	9	7	5	3

Table 3.1 - Standard f-numbers

3.3.4.1 Lens focus

The depth of field, or area of sharp focus, is dependent on aperture. There is a trade-off between aperture and exposure. A small aperture gives a large depth of field but requires a long exposure time. To reduce the exposure time the aperture must be increased or the sensor sensitivity increased. A large aperture reduces the depth of field. The depth of field for a different lens aperture can be calculated as follows:

If a lens is defocused, a point in the object is rendered as a small circle in the image, called the **circle of confusion**. The circle of confusion determines what is defined as in focus or out of focus. In the following calculations the circle of confusion is taken to

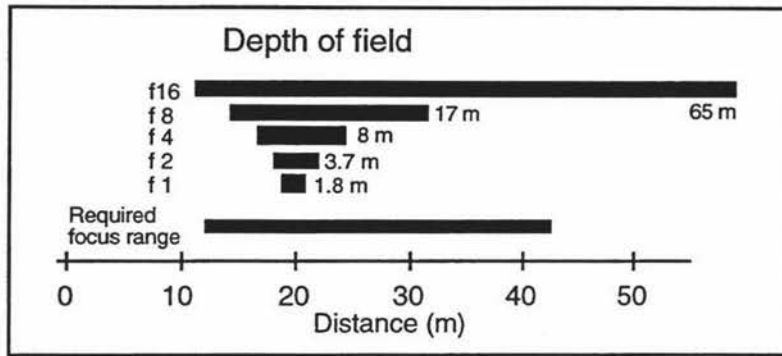


Figure 3.9 - Depth of field

be the size of a single element of the image sensor ($13 \mu\text{m}$ in the case of the TreeScan sensor). For 35 mm photography $30 \mu\text{m}$ is typically used.

Depth of field is defined by the following relationship, where S_{close} and S_{far} the near and far point of sharp focus, and S_o is the object distance :

$$\text{Depth of field} = S_{\text{far}} - S_{\text{close}}$$

$$S_{\text{close}} = \frac{h \times S_o}{h + (S_o - f)} \quad S_{\text{far}} = \frac{h \times S_o}{h - (S_o - f)}$$

$$h = \text{hyperfocal distance} = \frac{f^2}{N \times c}$$

For a 75 mm lens focused at 20 m figure 3.9 shows the depth of field for standard apertures two f-stops apart.

If an aperture of $f 1.4$ is used with the TreeScan system this provides a depth of field of 2.5 m when focused at 20m. This is sufficient provided the lens is refocused eleven times during the scan. At $f 4$ the same conditions provide a depth of field of 7.6 m and three refocusses during the scan are adequate.

3.3.4.2 Lens diffraction

Diffraction sets the maximum resolving power of a lens. When light from a point source passes through a narrow aperture it spreads out into a circle, or airy disc. When the diameter of the airy disc equals the circle of confusion the lens is said to be diffraction limited, and has reached the limit of its resolving power (Jacobson, 1993).

The diameter of the airy disc's first zero crossing can be shown to be:

$$\text{diameter 1st airy disc} = 2.44\lambda \frac{v}{d}$$

where λ is the wavelength of the light, v the distance of the image from the lens, and d the effective aperture. Approximating d to the lens focal length f , and using the fact that aperture has diameter f/N it follows that the diameter of the first airy disc is:

$$\text{diameter 1st airy disc} = 2.44N\lambda$$

Assuming a circle of confusion of 13 μm and typical wavelength of green light of 555 nm the lens diffraction limit can be calculated to be $f9.6$. No aperture smaller than $f9.6$ should be used. At $f9.6$ the depth of field is 13.8 to 36 meters.

$$\text{diffraction limit} = N = \frac{0.013}{2.44\lambda} = f9.6$$

3.3.4.3 Lens aberration

Lenses vary in quality due to lens aberrations. Lens aberrations are image defects that result from the limitations in the way lenses can be designed. Aberrations can never be eliminated, only reduced. A lens can have the following aberrations (Jacobson, 1993):

- **Spherical aberration** - Light passing through the edge of the lens is focused at a different distance than light striking near the centre.
- **Coma** - Light passing through the edge of the lens focuses in a ring displaced radially from the point where the light passing through the centre is focused.
- **Astigmatism** - Off axis points are focused at different distances in their radial or tangential direction.
- **Curvature of field** - Points in a plane get sharply focused on a curved surface.
- **Distortion** (pincushion and barrel) - The image of a square object has sides that curve in or out.
- **Chromatic aberration** - The position of sharp focus varies with wavelength.
- **Lateral colour** - The magnification varies with wavelength.

Blur caused by all aberrations except distortion and lateral colour can be reduced by using a small aperture. Conversely with a large aperture a lot of aberrations will be introduced into the image. Aspherical lenses minimise lens aberrations but are very expensive. The effects of pincushion or barrel distortion is most significant to the TreeScan system as it could affect the estimates made. Ideally the captured images should have no distortion but if quantified the distortion may be corrected for.

3.3.4.4 Modulation transfer function and relative illumination

The modulation transfer function provides an overall measure of lens performance that compares remaining modulation in the image plane with that of the original object as a function of spatial frequency. The result is expressed in percent, as a function of spatial frequency in line pairs per millimetre. As the spatial frequency increases the modulation transfer function and contrast level at which the lines are resolved decreases.

All images from photographic lenses vary in intensity from their centre to the edge. This is called relative illumination. There is a natural decrease from the centre to the outer edge which varies to the fourth power of the cosine of the field angle. The second major factor is light being blocked by mechanical vignetting. The effect of vignetting can be reduced by using a smaller aperture.

The modulation transfer function and relative illumination of a typical lens are shown in figure 3.10. Neither modulation transfer function or relative illumination is critical to the TreeScan development as absolute image intensity values are not used to estimate tree dimensions from captured images.

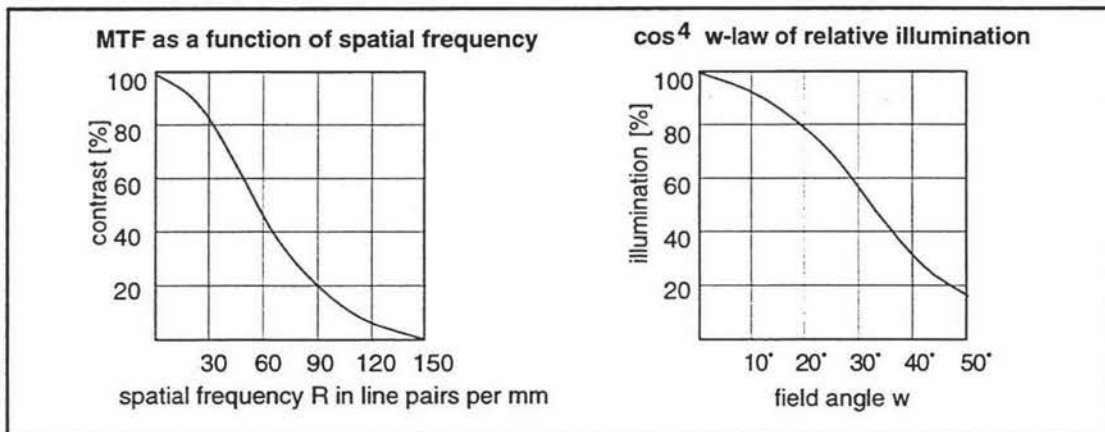


Figure 3.10 - Modulation transfer function and relative illumination

The availability of lenses was investigated. Key features important to the choice of lens were; motorised controls, C mount, and one inch format. Lenses were available from a series of manufacturers. The lens implemented in the TreeScan scanner was a 12 - 75 mm, f 1.8 - 360 Cosmicar TV zoom lens with motorised focus and zoom control and electromechanical aperture control. In the Mk2 model a manual fixed focal length (75 mm) C mount lens was implemented with a maximum aperture of f 1.4.

3.3.5 Image Focus

If a small aperture system is used there is sufficient depth of field to capture a single image without adjusting the focus. If however a large aperture system is used to obtain more light, focus adjustments will need to be made during the image capture. This should be implemented using some form of autofocus algorithm.

In order to set up an autofocus algorithm, a suitable measure of focus is required. The assumption that well focused images contain more information than unfocused images provides the basis for the criterion functions used by many autofocus systems. The criterion functions can be classified as; frequency domain functions, gradient functions, information content functions, and grey level variance (Groen et al, 1985).

Yeo et al (1993) evaluate four criteria functions for autofocusing in tissue microscopy that were selected for their computational simplicity and literature recommendation. The functions evaluated were the Tenengrad function, squared gradient function, Brenner function, and variance function. The results indicated all functions provided a good measure of focus.

The Brenner function, which was implemented in the TreeScan system, is a simple criterion that is gradient related. The difference in grey level intensity is taken between pixels two pixels apart, squared, and summed over the focus area.

$$f(l) = \sum_x \sum_y [l(x+2, y) - l(x, y)]^2$$

During the image capture the distance from the lens to the object being imaged changes, so the focus position must be adjusted. Three approaches can be adopted to retain focus throughout a scan:

- A single focus compromise can be set halfway up the tree.
- An autofocus may be made at several places during a scan.
- Focus during the scan can be calculated from one autofocus at the bottom and calculated geometry.

The relative advantages of these approaches is further discussed in section 5.2.5 of the TreeScan software chapter.

3.4 Scanner Interface and Image Storage

The images captured by the system under development are very large, up to eight megabytes per image. As a result it becomes important to consider the scanner interface that will be used to transfer images to computer and the image storage requirements once the image has been transferred to computer.

3.4.1 Scanner Interface

There are a large number of interface methods that can be used to transfer data between a computer and an external peripheral device. The interface method used must be able to provide acceptable data transfer rates while retaining the flexibility to provide computer control over the scanner.

The important difference between different interface methods is in transfer rates. Acceptable data transfer rates are determined by the time it takes the scanner to capture an image. For a fixed size image, as acquisition time decreases the data transfer rate required increases proportionally. In practise for an 8 megabyte image and an acquisition time of one minute the data rate required is 133 kByte / s. The table below shows the data transfer rate required for a variety of acquisition times:

Image transfer time	Data transfer rate
10 seconds	800 kByte / s
30 seconds	267 kByte / s
1 minute	133 kByte / s
5 minutes	26 kByte / s

Table 3.2 - Image acquisition time vs. data transfer rate

The interface between scanner and computer can be either analog or digital. Analog methods such as framegrabber cards provide high speed methods to transfer image data to the computer but are inflexible in the control they provide over the scanner. Digital methods provide more flexibility but are more restrictive for the high speed transfer of image data. Table 3.3 summarises the different interface methods available.

Frame grabber cards are able to capture information at video rates and transfer the information to the computer at high speed using the computer bus. Frame grabber cards are not available for portable computers and do not provide the required degree of flexibility.

Interface method	Data transfer rate	Flexibility
Analog		
Frame grabber	video rate 5 - 10 MByte / s	Low
Digital		
SCSI	4 MByte / s	High
Audio input	44 kByte / s (16 bit)	Low
Apple talk (serial RS422)	28.8 kByte / s	High
Serial	19.2 kByte / s	High

Table 3.3 - Scanner interface methods

A variety of digital interface methods are available on a standard portable computer. It was decided to use the high speed digital SCSI communications interface to implement the data transfer between the scanner and computer as this provided more than adequate speed with very high flexibility in scanner control.

3.4.2 Image Storage

Image storage is an important consideration as the images captured are very large. Both temporary storage during processing and long term storage for image archiving must be considered.

The images are captured straight on to computer and stored on hard disk. At eight megabytes per image a maximum of twenty images will fit on a typical 160 megabyte hard drive. At two images per tree this relates to ten imaged trees. In the short term a large (1-2 GByte) hard disk drive may need to be used for storage of images before processing.

Using the MARVL system a typical plot to be inventoried will contain about 15 trees to inventory. At a rate of 7 or 8 plots per day, this relates to 100 trees per day or 2 Gbytes of storage!

Possible solutions for reducing this storage requirement are to process the images immediately, use image compression techniques, only keeping the section of the image with the tree in it, or vertical decimation by discarding horizontal lines.

It should be noted that video tape although a lossy storage medium is probably the most cost effective medium to store large quantities of image data. It is in this area that the video camera imaging method as discussed in section 2.2.1 would have significant advantages over a computer storage based technique.

3.5 Parameter Extraction

Once the images have been captured, the next task is to establish the relationship between image dimensions and the real world tree dimensions. This is the **parameter extraction**. Parameter extraction consists of image calibration and distortion correction:

- **Image calibration** establishes what dimensions a single pixel represents on the object being imaged at the calibration reference point.
- **Distortion correction** performs a mathematical correction, based on the calibration information, to compensate for the perspective distortions introduced for any point not at the calibration reference. Image calibration and distortion correction are closely related and will be discussed together.

Planar transformation or geometric distortion correction?

There are two fundamentally different approaches that can be taken to correct for perspective distortion:

1. The distortion can be seen as a **planar transformation**. Four points on the image and four points on the calibration rod uniquely identify the transformation. If the real world dimensions of the calibration rod are known, the position of any point on the tree can be calculated on the calibration plane.
2. The task can be seen as **geometric**. If the position of the scanner is known in relation to the calibration rod, and the angle of the tree plane is known, the position of any point on the tree can be calculated on the calibration plane.

Four distortion correction methods have been implemented, some based on planar

Correction method	Accuracy	Based on
Planar transformation correction		
Simple perspective correction	Accurate	Width of 2 cross bars & spacing (4 points in space)
TreeScan perspective correction	Approximate and imprecise	Width of 2 cross bars & spacing in two correction steps
Geometric correction		
TreeScan perspective correction	Imprecise	Width of 1 cross bar & calculated angle (using cross bars & spacing)
TreeScan perspective correction	Accurate	Width of 1 cross bar & 2 measured angles

Table 3.4 - Comparison of distortion correction methods

transformation and some based on geometric correction. Each of these methods relies on different calibration information and has advantages and disadvantages. During the development precision problems were encountered when relying solely on image calibration rod dimensions as calibration information. As a result the calibration procedure was modified and additional angular information measured. All distortion correction methods are defined in this chapter with further discussion on the reasons for the final implementation in section 6.1.

3.5.1 Image Calibration

Image calibration involves having an object of known size (calibration rod) in the image. Using the dimensions of the calibration rod in the image, the dimensions of other objects the same distance from the scanner can be calculated. This will allow the size of features near the bottom of the tree to be estimated. The size of features near the top of the tree cannot be estimated this way because a calibration rod as tall as the tree would be required.

It should be noted that in image processing, image calibration can be based on either linear measurement or area measurement. Image calibration based on linear measurement will provide a scaling factor of 'size per pixel' in the direction of the estimate. Calibration based on area measurement will provide a scaling factor in both x and y direction. Calibration based on area measurement has the advantage that it can be more precise and is resistant to image noise, but it is slightly more difficult to implement.

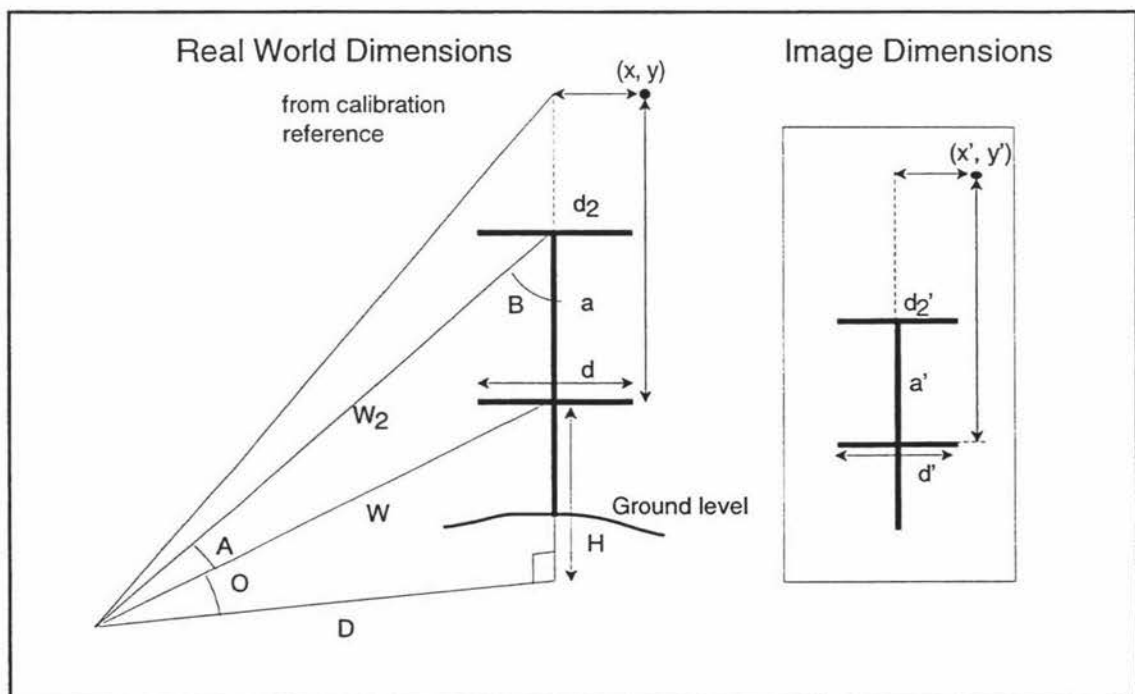


Figure 3.11 - Definition of terms

Depending on the approach taken to correct for image distortion different calibration information is required. Different types of calibration rod have been made to supply this information.

The planar transformation distortion correction requires four fixed points and a calibration reference. A calibration rod with a vertical pole and two cross bars was used to generate this information.

The calibration information required by geometric distortion correction with measured angle O is : a calibration reference, distance from this reference, and the combined dip / lean angle. In this case a smaller calibration rod is adequate. A single cross bar can be used to estimate distance with the centre as zero reference. A further improvement that can be made is to use a circular object instead of the single cross bar to use image calibration based on area measurement.

The calibration method used for the final TreeScan system is to:

- Have both a cross bar and calibration circle to provide two ways of estimating distance, with the intention of using only the circle in the future.
- Measure the dip angle from the scanner to the calibration reference, and the angle of tree lean while capturing the second image at right angles.
- Use centre of the circle / middle of the crossbar as the calibration reference.

Both tree lean and dip angle as a result of elevated or lowered scanner position are combined into one angle O (see figure 3.17 for explanation).

3.5.2 Planar Transformation Distortion Correction

The perspective distortion introduced during image capture can be interpreted as a planar transformation. Four points on the image and four points on the calibration rod uniquely identify the transformation. If the real world dimensions of the calibration rod are known the position of any point on the calibration plane can be estimated.

3.5.2.1 Simple perspective correction

If an area camera is used for image capture, the distortion introduced is a linear distortion from rectangular space to triangular space. This will be referred to as a simple perspective distortion. A planar transformation can be used to convert back from the triangular space to the rectangular space based on four points in the space. The four points used are the ends of the calibration crossbars. This is the correction method implemented for the macros developed for the experimental photographic system (Pugmire, 1994).

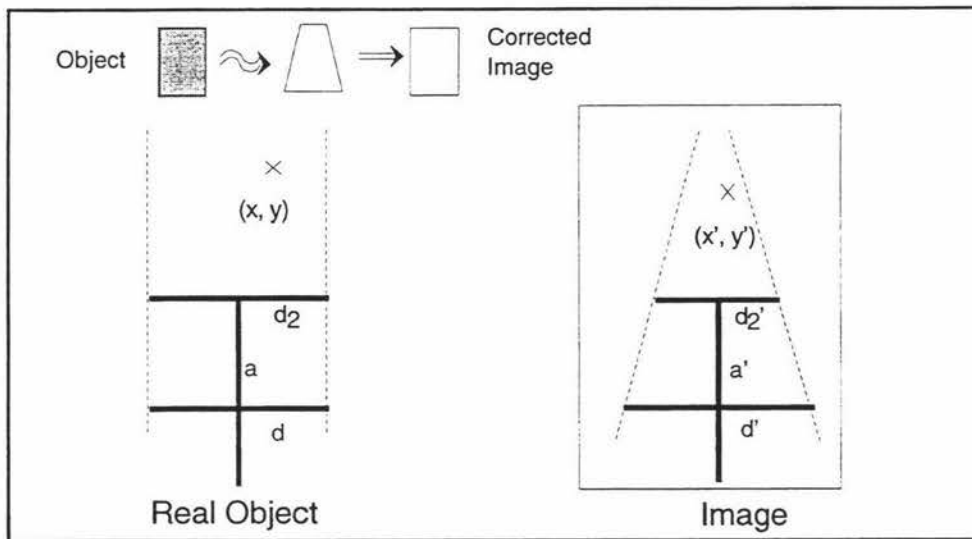


Figure 3.12 - Simple perspective correction

The positions of the calibration rod end points are established using the real dimensions of the calibration rod (crossbar width d and d_2 , and distance apart a) and the image dimensions of the calibration rod (crossbar widths d' and d_2' , and distance apart a'). The real world x and y coordinates are calculated from the image coordinates using the following equations:

$$y = \frac{y'a(1-P)}{a'-Py'} \quad x = \frac{x'd}{d'} \left(\frac{a}{a'-Py'} \right) \quad \text{where} \quad P = 1 - \frac{d_2'}{d'}$$

3.5.2.2 TreeScan perspective correction (approximation)

If the TreeScan camera is used for image capture, the distortion will become nonlinear. This distortion (explained in section 3.3.3) will be referred to as the TreeScan perspective distortion. Four points still uniquely define the transformation, however image lines represent equal angular step sizes. The consequence of this is that pixels near the top of the TreeScan image will represent larger distances on the equivalent area scan image as shown in figure 3.13.

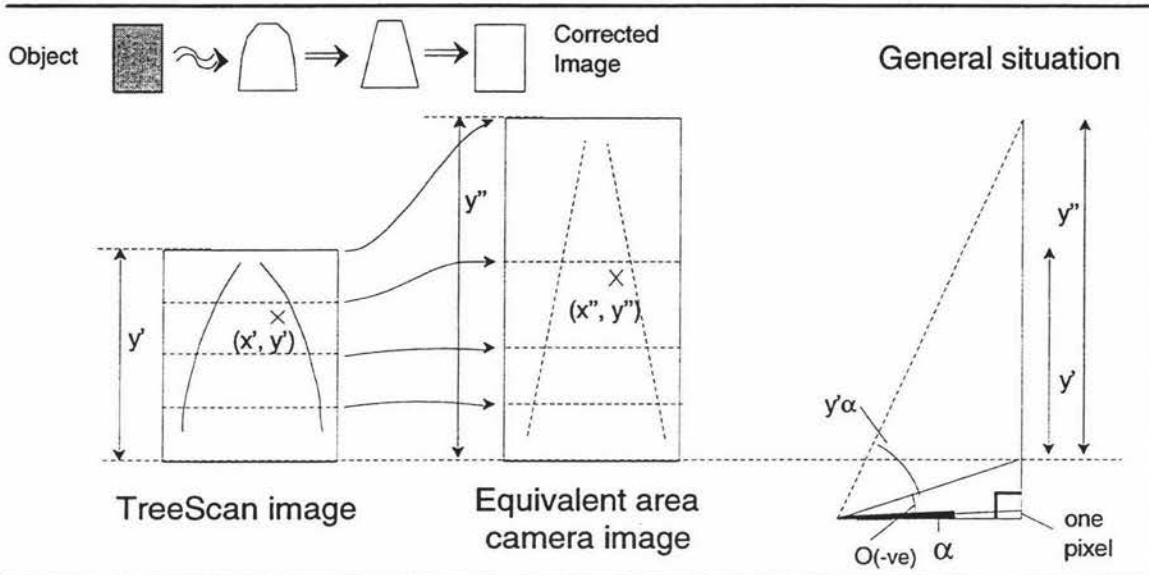


Figure 3.13 - Two step perspective correction

This distortion can be corrected for in a two step process. Coordinates on the TreeScan image can be converted to equivalent coordinates within an area camera image using an angular correction. These coordinates are then processed by the *simple perspective correction* method described in the previous section.

$$y'' = \frac{\tan(y'\alpha - O)}{\tan \alpha} \quad \alpha = \text{Step angle}$$

$$y = \frac{y''a(1-P)}{a' - Py''} \quad x = \frac{x'd}{d'} \left(\frac{a}{a' - Py''} \right) \quad \text{where } P = 1 - \frac{d_2'}{d'}$$

If the principal scanner axis is not normal to the calibration plane (the angle O is zero), this two step correction method becomes an approximation.

The second restriction of this method is that it suffers from the precision problems discussed in the section on geometric correction. Given these restrictions the usefulness of this correction method is limited.

3.5.3 Geometric Distortion Correction

The distortion correction task can also be interpreted as a geometric correction. This involves two steps; the scanner position relative to the calibration rod is determined to establish the image capture geometry, which is then used to estimate the position of any point on the calibration plane.

3.5.3.1 TreeScan perspective correction - derived O (imprecise)

Once the scanner position in relation to the calibration rod is known the coordinates of any position on the tree can be easily estimated using H , D and O . The y coordinate can be estimated using elementary trigonometry, and x is calculated using the fact that scaling in the x direction (or magnification) is inversely proportional to distance. To get the x coordinate, x' is scaled by the calibration factor d/d' and then by the ratio of distances W_i/W .

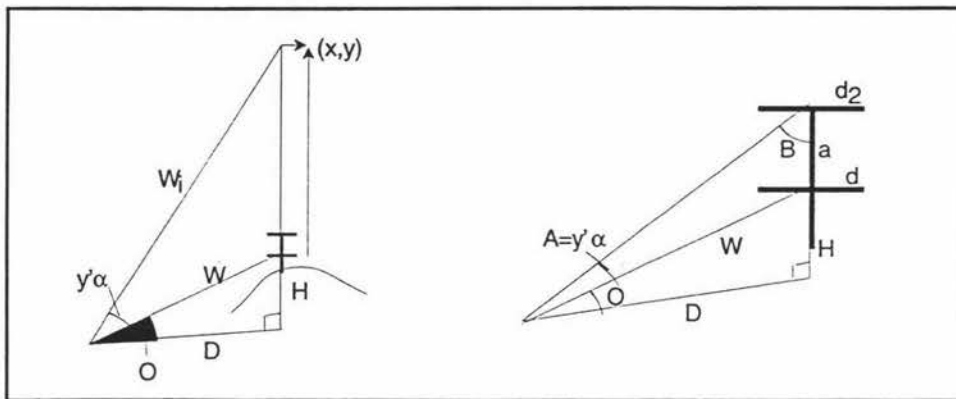


Figure 3.14 - Geometric correction using derived O

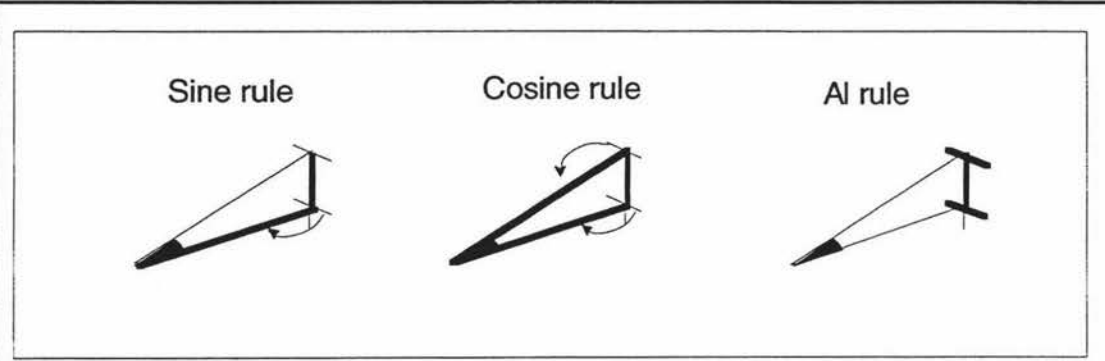
$$y = \tan(y'\alpha + O) \times D - H \quad x = \frac{x'd}{d'} \left(\frac{\cos(O)}{\cos(y'\alpha + O)} \right)$$

The scanner position relative to the calibration rod can be calculated from the calibration rod dimensions in the image. This involves estimating distance W from the calibration reference using the image width of the lower calibration bar, and the calculation of the angle O from other image dimensions of the calibration rod.

The situation is slightly redundant and different methods may be used to calculate O from the calibration rod dimensions. Three methods implemented were the *sine rule method*, *cosine rule method*, and the *AI method*¹ as described in figure 3.15.

However each of these methods suffers from a precision problem as a result of using a parameter to a degree of accuracy an order of magnitude greater than it can be measured from the image.

¹ Thanks to Alistair Hall for help with the derivation of the mathematics for this method.



Sine rule method - This method to calculate O is based on the sine rule.

The distance from the calibration reference can be estimated using the image width of the bottom cross bar, the angle A can be measured directly from the image using the image calibration rod height, and a (crossbar height) is physically measured. Using the sine rule O can be calculated.

$$O = 90 - A - B \quad A = a' \alpha \quad B = \sin^{-1} \left(\frac{W \sin A}{a} \right)$$

Cosine rule method - This method to calculate O is based on the cosine rule.

The distance from the calibration reference can be estimated using the image width of the bottom calibration rod cross bar, the distance from the top of the calibration rod can be estimated using the image width of the top calibration rod crossbar, and a is measured. Using the cosine rule O can be calculated.

$$O = 90 - A - B \quad A = a' \alpha \quad B = \cos^{-1} \left(\frac{a^2 + W_2^2 + W^2}{2aW} \right)$$

Al rule method - This method is based on solved simultaneous equations describing the situation. Corrected x and y coordinates are directly calculated without the intermediate step of calculating O .

The image widths of both the top and bottom cross bars are used, and the angle A is measured directly from the image using the image calibration rod height.

$$O = \tan^{-1} \left(\frac{\cos(a' \alpha) - \frac{d_2'}{d'}}{\sin(a' \alpha)} \right) \quad y = a \frac{\cos(a' \alpha + O)}{\sin(a' \alpha)} \frac{\sin(y' \alpha)}{\cos(y' \alpha + O)}$$

$$x = \frac{x' d}{d'} \left(\frac{\cos(O)}{\cos(y' \alpha + O)} \right)$$

Figure 3.15 - Correction based on calibration rod dimensions

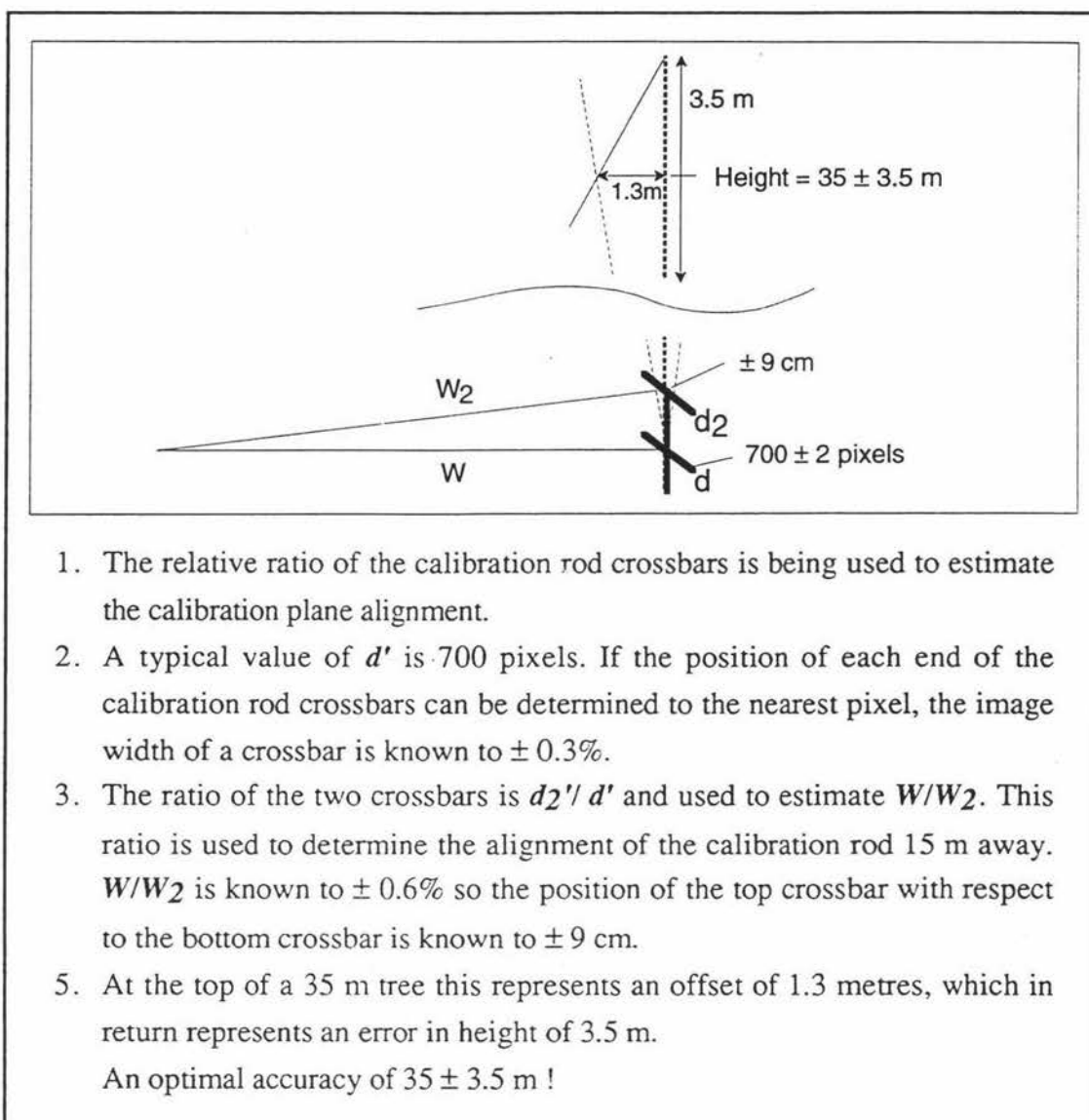


Figure 3.16 - Distortion correction imprecision

When estimating the scanner position what is effectively being done is the estimation of the alignment of the calibration plane (or rod) with respect to the scanner. It is the projection on this calibration plane of objects that is being measured, so in order to estimate the tree dimensions accurately, the positioning of the calibration plane must be known to a much higher degree of accuracy than the dimensions being estimated.

If calibration rod points can be located on the image to 1 pixel accuracy this represents an optimal height accuracy of ± 3.5 m at a height of 35 m (as shown in figure 3.16). In order to identify heights to a ± 10 cm accuracy the calibration rod points need to be identified to a subpixel accuracy of 0.03 pixel. An alternative is to find a different method to determine the angle θ .

3.5.3.2 TreeScan perspective correction - measured O

This final distortion correction method is again based on the geometric correction that position coordinates can be calculated using known H , D and O . The same equations used in the previous section are used to estimate these coordinates.

$$y = \tan(y' \alpha + O) \times D - H \quad x = \frac{x' d}{d'} \left(\frac{\cos(O)}{\cos(y' \alpha + O)} \right)$$

where $O = \text{Dip} + \text{Tree lean}$

The angle O is however physically measured rather than derived from image calibration rod dimensions. By measuring O the imprecision experienced by the previous methods that derive O can be eliminated.

The angle O is the combined angle of tree lean and dip as a result of a scanner position not level with the calibration rod. As a result the angle O is the sum of the tree lean measured from the vertical and the measured dip between the horizontal and the principal axis. O can be measured directly to the required degree of accuracy.

The height deviation of ± 3.5 m discussed in the previous section is equivalent to an accuracy in angle measurement of 2.2 degrees. Using a digital builders' level each of dip and lean can be measured to ± 0.1 degrees. This translates to an accuracy in height measurements of ± 30 cm. This is the method implemented in the Mk2 version of the TreeScan system.

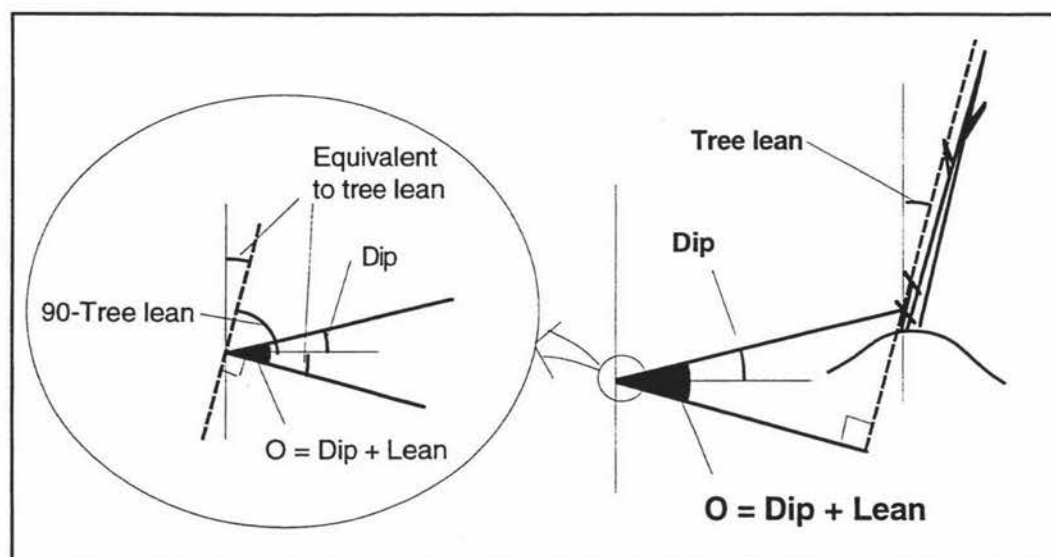


Figure 3.17 - Measurement of angle O

Note: A final comment that should be made is that the centre of the image should be used as the reference for x coordinates in this distortion correction as this is the only part of the image that remains horizontally undistorted.

3.6 Three Dimensional Model Generation

Three dimensional model construction must combine stem shape information from multiple views to form a three dimensional model of the tree stem.

To obtain a three dimensional model of a tree a minimum of two views at right angles is required. To ensure all branches have been observed a minimum of two views at 180 degree spacing is required. Both a view of all branches and a three dimensional model for the assessment of sweep can be obtained from three views at 120 degree spacing. If only a sample of branches is required then two views at right angles is sufficient.

The TreeScan software has been designed to estimate tree shape, and the size of visible branches from two views at right angles.

The three dimensional model consists a series of stem diameter and stem position estimates along the length of the tree stem. The model should consist of sufficient 'slices' to accurately determine any shape changes in the stem.

Stem diameter and stem position estimates are obtained for the two edges of the stem. By combining edge information of two views at right angles a rectangle is defined within which the stem lies. It must be assumed that the centre of the rectangle represents the centre of the tree stem and that the stem diameter is the magnitude of an inscribed ~~circle~~ ^{ellipse} as shown in figure 3.18.

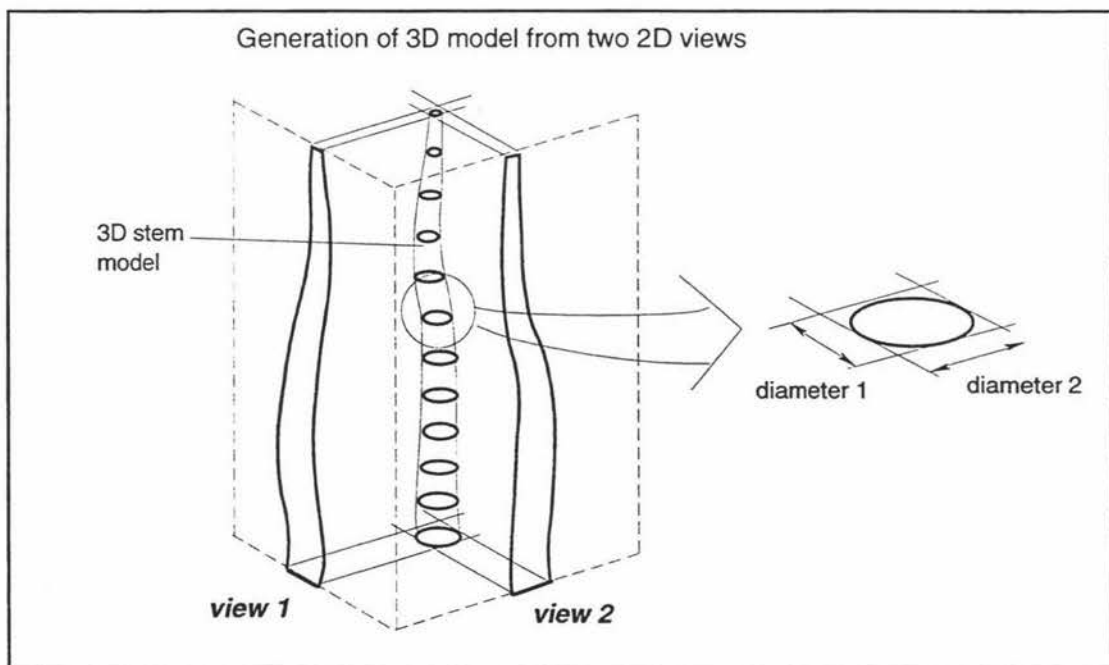


Figure 3.18 - Three dimensional model generation

Some trees will have a stem that is slightly oval in shape rather than circular. The two estimated diameters may or may not reflect this. If the major axis of the oval lies in the axis of one of the views the two diameter estimates will be diameter estimates along the major and minor axis of the tree stem. If the major axis does not lie in the axis of one of the views, both estimated diameters may be the same even though the tree stem is oval.

It is important to note that height information is duplicated as it is available in both views. If a calibration error is introduced and the calibration plane does not lie exactly within the tree an expected height error is introduced in estimates (see section 3.7). This means that there will be a discrepancy between height information from the two views. By combining height information from the second image with that of the first image a more accurate height estimate could be made. This may be done in two ways:

1. The height estimates of a feature common to both images could be compared and an indication could be gained of the accuracy of calibration. By adjusting other height estimates so that this particular height estimate matched in both views the average error could be reduced. This would require points to be marked in both images.
2. Height estimate may be corrected slightly by modifying the calibration with the feature distance in front of or behind the calibration reference from the second view. This would provide a unique adjustment for each point on the stem and account to a certain extent for tree shape variation. Feature size estimation would become an iterative process which requires the marking of the full stem shape first.

At this stage these modifications have not been made as the system is still undergoing testing. The improvement these two methods would make to the accuracy of the system is expected to be minimal.

3.7 Implications of Image Capture Geometry

The TreeScan system must be able to make accurate estimates from images captured under poor geometric image capture conditions, which may introduce a series of errors in the tree size estimates. These errors will be referred to as **expected errors** as their magnitude can be calculated. This section discusses the various sources of expected error and calculates their significance to the accuracy of the TreeScan system.

The image capture geometry is poor because calibration information must be extrapolated. Images can only be captured and calibrated at ground level and must typically be captured in the range of ten to twenty metres from the base of the tree, this is drawn to scale in figure 3.19. The images are calibrated at the base of the tree providing good precision there.

To estimate dimensions near the top of the tree the calibration information must be extrapolated using the distortion correction methods. As a result errors will be introduced. In addition to this, for estimates made at an oblique angle to the calibration plane (height estimates towards the top of the tree) the expected error will be accentuated.

This image capture geometry is inherent in in-field tree imaging and cannot be improved upon without a conceptually different approach to the estimation of tree parameters.

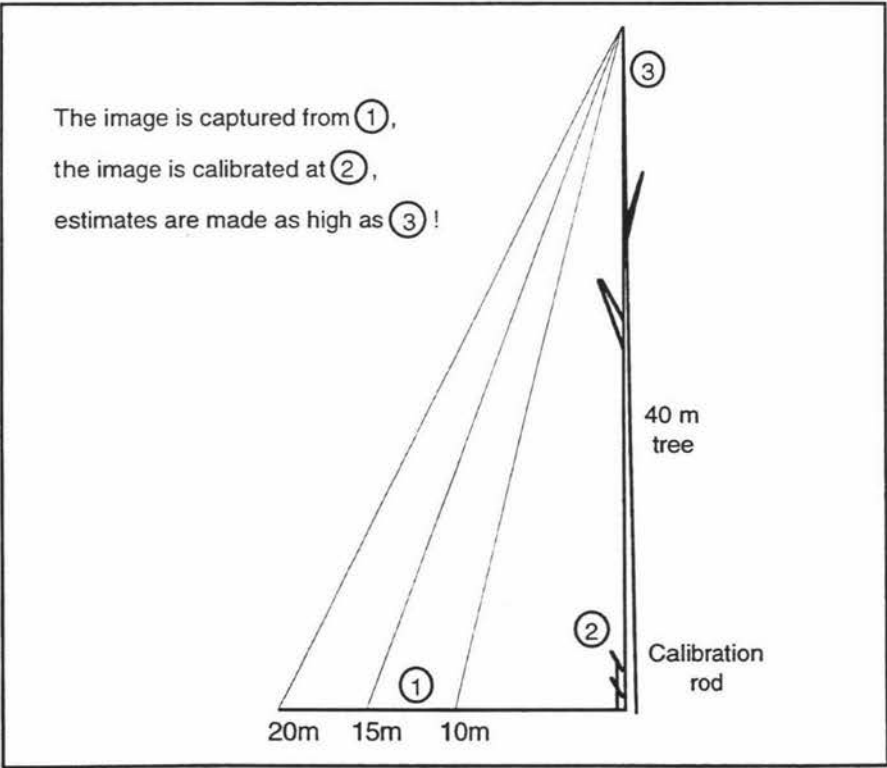


Figure 3.19 - Image capture geometry (to scale)

The effect of the image capture geometry is different on the various size estimates:

- **Branch diameters**, stem diameters, and feature sizes are estimated perpendicular to the calibration plane and the introduced error will be small.
- **Height estimates** are made at an oblique angle to the calibration plane and introduced errors will be larger.

The expected errors in the TreeScan estimates can be divided into three sources:

1. Tree plane variation
2. Variation in calibration data
3. Image processing and feature marking precision

Table 3.5 summarises the effects from each of these sources under typical operating conditions, which are taken to be: the estimation of parameters at a height of 30 m, and image capture distance of 15 m from the base of the tree. The magnitude of the expected error classified as: major ($>0.5 \times$ required specifications), minor ($0.2 - 0.5 \times$ specifications), and insignificant ($<0.2 \times$ specification).

Brief discussion of each relevant source of uncertainty is provided in sections 3.7.1 to 3.7.3, with further calculations provided in appendix E.

Cause of uncertainty	Significance of expected error		
	Height	Stem diam.	Branch diam.
Tree plane variation			
Tree displacement	Major	Insignificant	Insignificant
Calibration alignment variation			
Variation in measured angle	Major	Minor	Insignificant
Calibration rod alignment	Minor	Minor	Minor
Calibration rod vertical placement	Minor	Insignificant	Insignificant
Calibration rod in front of tree	Insignificant	Minor	Minor
Image processing precision			
Calibration end pixel placement	Minor	Minor	Minor
Calibration centre point placement	Minor	Insignificant	Insignificant
Pixel placement on feature	Insignificant	Minor	Major

Table 3.5 - Sources of expected error in TreeScan

3.7.1 Tree Plane Variation

If a section of tree is not straight (as a result of lean, sweep, or kink) so that the feature of interest is offset from the calibration plane there will be a tree displacement, introducing expected errors. For tree lean the error can be eliminated by aligning the scanner correctly, however for a tree with sweep or a kink this may not be possible and a compromise will need to be made in aligning the scanner.

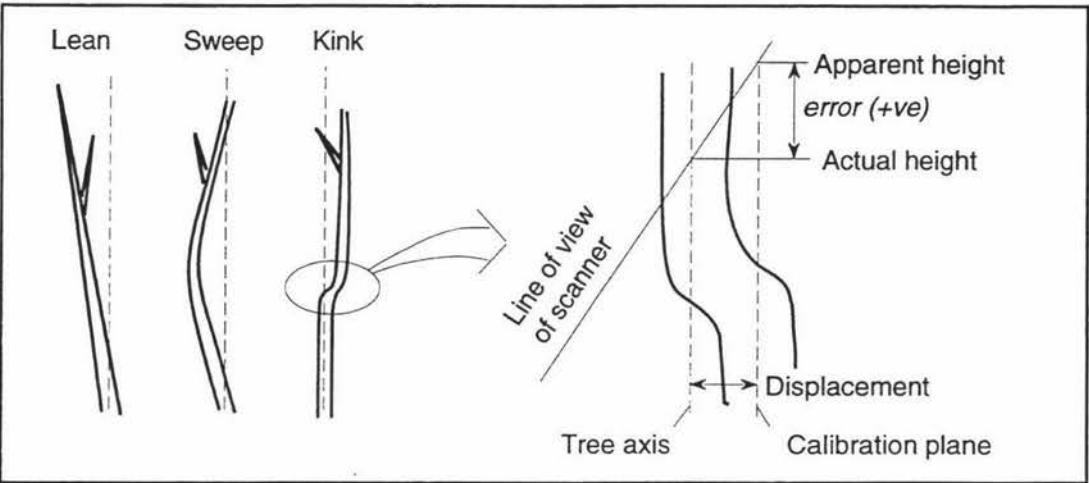


Figure 3.20 - Tree plane variation

The software can be extended to use calibration information from the second view to modify the calibration information of the first view. This would reduce the magnitude of introduced errors, but implies any height must be iteratively refined by processing both views (see section 3.6).

3.7.1.1 Errors Introduced by Tree Displacement

Tree displacement can cause a major error in height estimates, as any error is accentuated by the geometry. As shown in table 3.6, a stem displacement of only 50 cm will cause a height to be overestimated by 1.03 m for a height estimate at 30 m from an imaging position 15 m away from the tree (see appendix E for more detail).

Height error (m)		Height of tree estimates (m)				
Dist from tree (m)		0	10	20	30	40
	Height error (m) introduced by 0.5 degree error in θ					
10		0.00	0.53	1.05	1.58	2.11
15		0.00	0.34	0.69	1.03	1.38
20		0.00	0.26	0.51	0.77	1.03

Table 3.6 - Height errors introduced by stem displacement

Width errors are not as badly affected by stem displacement as width estimates are based solely on distance from the feature. For the same situation described above the error in diameter estimates will be 3 mm for a 10 cm branch (see table 3.7).

Width error (cm)

Dist from tree (m)	Height of tree estimates (m)				
	0	10	20	30	40
	Width error (cm) from 50 cm stem displacement (10 cm branch)				
10	-0.5	-0.5	-0.5	-0.5	-0.5
15	-0.3	-0.3	-0.3	-0.3	-0.3
20	-0.2	-0.2	-0.2	-0.2	-0.2

Table 3.7 - Width errors introduced by stem displacement

The introduction of these expected errors can be minimised by ensuring the scanner and the calibration rod are closely aligned with the tree during image capture.

Note that in all error tables (3.6 to 3.9) the error values for typical conditions, estimates at a height of 30 m imaged 15 m away from the tree, are highlighted in bold.

In short:

- The variation in tree plane can introduce a major error in height estimates. Any errors in height estimates will be accentuated near the top of the tree.
- The variation in tree plane can introduce a minor error in diameter estimates. Diameter estimate errors remain constant up the tree.

3.7.2 Calibration Alignment Variation

The image can only be calibrated to the degree of accuracy that the calibration information can be measured or estimated. This section discusses the effects of imprecise calibration information and calculates the degree of accuracy to which the

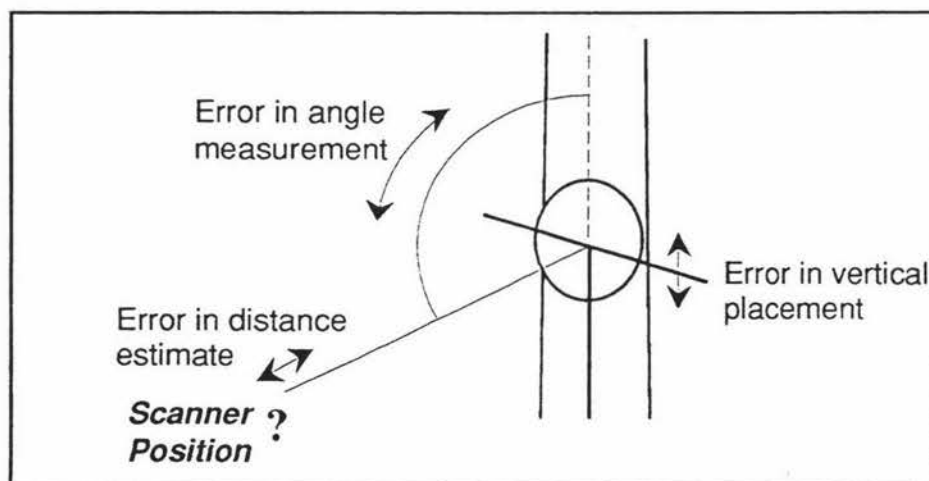


Figure 3.21 - Calibration alignment variation

calibration information needs to be measured. Any imprecision in calibration alignment of the Mk2 system can be divided into three sources; variation in measured angular position, variation in distance estimate, and calibration rod placement.

Analysis of the variation in the calibration alignment of the Mk1 system was more complicated (see Weehuizen 1994c for a discussion).

3.7.2.1 Errors Introduced by Variation in Measured angle

If the angular position of the scanner w.r.t. the tree (angle *O*) is inaccurately measured or not entered to calibrate the image, errors are introduced into the size estimates. Inaccurate measurement of *O* affects the 'alignment' of the calibration plane to which dimensions are being estimated (see table 3.8).

- Inaccurate measurement of *O* can introduce a major error in height estimates. Any errors in height estimates will be accentuated near the top of the tree.
- Inaccurate measurement of *O* does not introduce significant errors in width estimates.

The angle *O* should be measured to an accuracy of ± 0.2 degrees. This provides a precision in the height estimates of ± 20 cm for estimates taken at a tree height of 30 m from images captured at a position 15 m from the base of the tree.

Height error (m)					
Dist from tree (m)	Height of tree estimates (m)				
	0	10	20	30	40
	Height error (m) introduced by 0.5 degree error in O				
10	0.00	0.09	0.36	0.81	1.45
15	0.00	0.06	0.24	0.53	0.95
20	0.00	0.04	0.18	0.40	0.71
Width error (cm)					
Dist from tree (m)	Height of tree estimates (m)				
	0	10	20	30	40
	Width error (cm) introduced by 0.5 degree error in O				
10	0.0	-0.2	-0.1	-0.1	-0.1
15	0.0	-0.2	-0.2	-0.1	-0.1
20	0.0	-0.1	-0.2	-0.2	-0.1

Table 3.8 - Errors introduced by variation in measured angle

3.7.2.2 Errors Introduced by Calibration Rod Alignment

The calibration distance is the distance from the point the image was captured to the calibration reference. The width of the calibration rod cross bars is used to estimate this calibration distance. If the calibration rod is not at right angles to the direction of image capture an error is introduced into the estimated calibration distance.

A distance error of 25 cm (see table 3.9) implies the calibration rod has rotated by 10 degrees (or each end of the calibration rod has moved by 18 cm).

- Inaccurate measurement of calibration distance can introduce a minor error in height estimates. Errors in height estimates are accentuated near the top of the tree.
- Inaccurate measurement of calibration distance does not introduce significant errors in width estimates.

The calibration rod should be perpendicular to the principal axis to within ± 6 degrees (crossbar ends can move up to ± 11 cm). This estimates the calibration distance to ± 10 cm and provides a height estimate precision of ± 20 cm for estimates taken at a tree height of 30 m from images captured at a position 15 m away from the base of the tree.

Height error (m)

Dist from tree (m)	Height of tree estimates (m)				
	0	10	20	30	40
	Height error (m) introduced by 25 cm error in Dist.				
10	0.00	0.25	0.50	0.75	1.00
15	0.00	0.17	0.33	0.50	0.67
20	0.00	0.13	0.25	0.38	0.50

Width error (cm)

Dist from tree (m)	Height of tree estimates (m)				
	0	10	20	30	40
	Width error (cm) introduced by 25 cm error in Dist.				
10	-0.2	-0.2	-0.2	-0.2	-0.2
15	-0.2	-0.2	-0.2	-0.2	-0.2
20	-0.1	-0.1	-0.1	-0.1	-0.1

Table 3.9 - Errors introduced by distance error

3.7.2.3 Errors Introduced by Calibration Rod Vertical Placement

The calibration rod is used as the calibration reference. The calibration rod must be placed at the same height in both images to retain the same height as a calibration reference. If the rod is not placed at the same height this introduces:

- Height estimate error of the size of the calibration rod displacement.
- No width estimate errors.

The calibration rod should be placed at the same height in both views to an accuracy of ± 5 cm. This provides a precision of ± 5 cm for height estimates

3.7.2.4 Errors Introduced by Calibration Rod Placement in Front of Tree

Trees are three dimensional and the stem has depth. The calibration rod cannot be placed directly in line with the centre of the tree, instead it must be placed in front of the tree. The result of this is that there is a small stem offset error (half stem diameter) introduced near the base of the tree. The effect of stem offset error is described in section 3.7.1.

- Calibration rod placement can introduce a minor error in height estimates.
- Calibration rod placement does not introduce significant errors in width estimates.

The calibration software can take this into account and correct both height and diameter estimates for this.

3.7.3 Image Processing and Feature Marking Precision

In order to calibrate the image and estimate feature sizes, points must be marked on the image. The placement of these points introduces imprecision as the marking relies on human judgement and the ability to see the feature of interest in the image. This section discusses the effects of imprecise image marking and calculates the degree of accuracy to which the points need to be marked.

3.7.3.1 Errors Introduced by Calibration Rod End Pixel Placement

The marking of the calibration rod end points provides information on their location in the image. Using these marked points and the known calibration rod width the calibration distance is calculated.

- Minor error introduced in height estimates
- No error introduced in diameter estimates

The end points should be marked to within ± 1 pixel. This estimates the calibration distance to a precision of ± 4 cm, which provides a precision in the height estimates as a result of calibration rod end pixel placement of ± 8 cm for estimates taken at a tree height of 30 m from images captured at a position 15 m from the base of the tree.

3.7.3.2 Errors Introduced by Calibration Rod

Centre Pixel Placement

The centre of the calibration rod is used as the calibration reference and taken to be at breast height. All estimates are made with reference to this. If the placement of this point is imprecise all height estimates will be out by several cm.

- Minor error introduced in height estimates
- No error introduced in diameter estimates

The centre of the calibration rod must be marked to within ± 1 pixel. This provides a precision in the height estimates as a result of calibration rod centre pixel placement of ± 1 cm for estimates taken at a tree height of 30 m from images captured at a position 15 m from the base of the tree.

3.7.3.3 Errors Introduced by Pixel Placement on Feature

The pixel placement on features whose size is being estimated provides their location information. Imprecise placement has the following effect:

- No significant error introduced in height estimates
- Major error introduced in diameter estimates

As diameter estimates are calculated using the differences in absolute position, and the two sides of a branch typically will be separated only by a diameter of several pixels estimates are very sensitive to precise feature marking.

The end points of the line selection for branch sizes should be marked to sub pixel accuracy if possible. This provides a precision in branch size estimates of a pixel resolution of ± 0.7 cm near the top of the tree.

Chapter 4

TREESCAN HARDWARE

4.1	TreeScan Hardware Overview -----	68
4.2	Scanner Hardware Overview -----	70
4.3	Microcontroller Subsystem -----	75
4.4	SCSI Subsystem -----	80
4.5	Line Scan Camera Subsystem -----	87
4.6	Additional Hardware -----	92
4.7	Hardware Development Environment -----	103

This chapter describes the hardware of the TreeScan system. The main focus is on a description of the custom designed scanner.

The scanner hardware is described by dividing the scanner into functional blocks. The interaction between these functional blocks is explained, then a number of aspects of each functional block are described in greater detail, including; operating theory, building blocks, schematics, reasons for the chosen implementation, and problems encountered during development.

4.1 TreeScan Hardware Overview

The TreeScan system is a complete system that consists of a portable computer, a custom designed scanner and a calibration rod. In addition to this there is a tripod on which to mount the scanner, a scope sight and digital level to align the scanner, a set of batteries to power the scanner, cables to connect the scanner to the computer and power supply, and a set of cases to house the entire system.

Normal operation of the TreeScan system involves two separate operations:

1. **Image Capture:** Images must be captured using the scanner under control of the portable computer. All system components must be taken into the forest to capture images.
2. **Parameter Extraction:** Tree dimensions must be estimated using routines implemented in software. The parameter extraction requires only the portable computer. This may be carried out at any time after the images have been captured, either in the forest or back in the office.

The system configured for **image capture** consists of the scanner set up on the tripod and connected to the portable computer, as shown in figure 4.1. The scanner is pointed at the tree to be imaged with the calibration rod fully extended and placed against the tree. The scanner is then aligned with the tree using the scope sight and its angular position with respect to the tree recorded. The image can then be captured.

The scanner captures the image data under control of the computer and passes the image data to the computer for storage. The computer sends high level *scanner control commands* (SCCs) to the scanner. The microcontroller inside the scanner carries out tasks based on these. Scanner control commands are high level instructions such as; move the scanning mirror home, capture a block of lines, or move the lens focus to infinity.

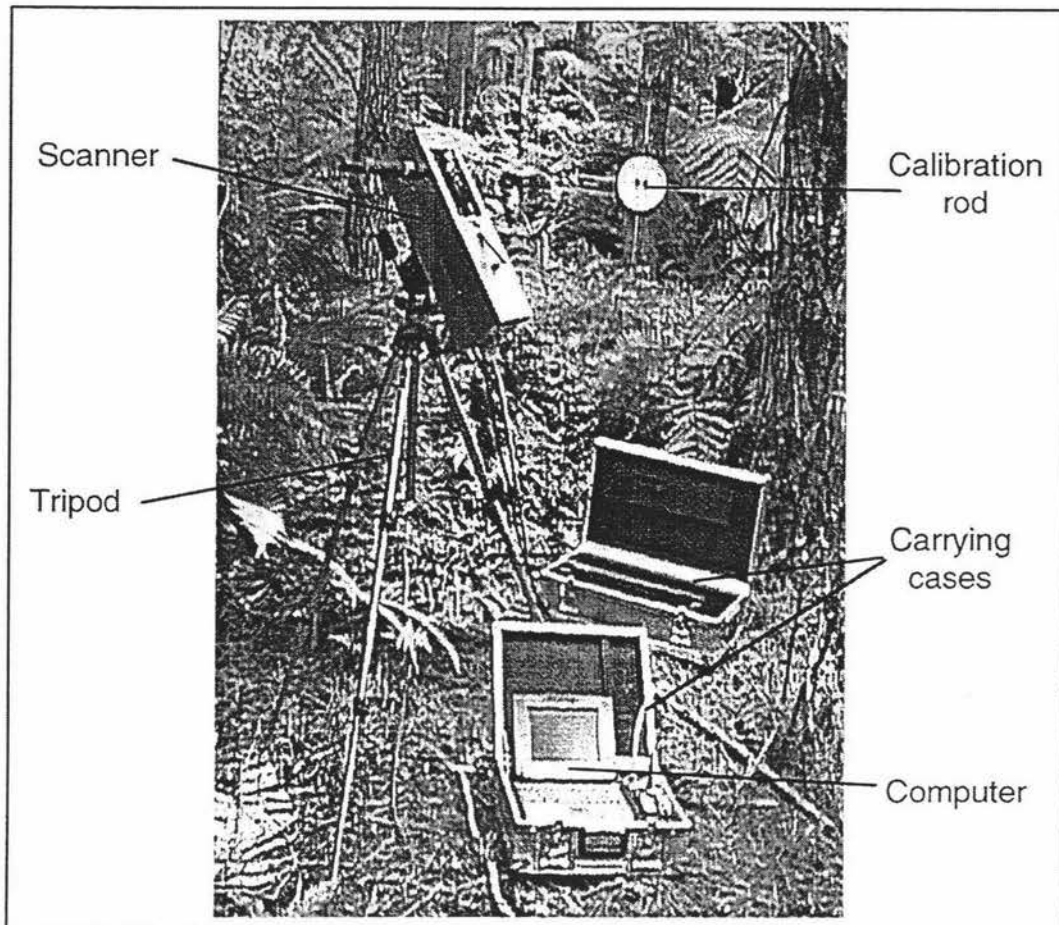


Figure 4.1 - TreeScan system ready for image capture

The portable computer and scanner batteries are permanently housed in the smaller of the two protective carrying cases (see figure 4.1). All other components including scanner, tripod, cabling, scope sight and digital level fit into the second protective case.

The portable computer is an Apple Macintosh Powerbook 520c with 20 megabytes of RAM. The Powerbook 520c is based on the Motorola 68040 processor running at 25 MHz. Combined with the flexible NIH Image image processing software^(see page 107), the Powerbook provides a powerful image processing environment. The Powerbook also has inbuilt support to connect high speed external SCSI devices.

The hardware and software are closely interrelated in any complex microprocessor based system such as the TreeScan system. The choice of hardware determines how the software is implemented and influences the flexibility of the system. The hardware implemented was chosen for its flexibility, relatively low cost, and to provide a short development time. Where the hardware has limitations, in some cases these can be circumvented by a resourceful software implementation.

4.2 Scanner Hardware Overview

The scanner is the "camera" in the system and is a custom designed scientific instrument. During an image capture there are a large number of time critical tasks to be coordinated. A dedicated microcontroller based instrument provides the ability to coordinate these tasks while retaining maximum control and flexibility.

A microprocessor based instrument can be designed so that essential data processing is handled by the microcontroller itself or by dedicated hardware (such as specialised A/D converters), which will be faster than the general purpose microcontroller hardware. However, the greatest flexibility will be maintained if the microcontroller hardware is used. For the TreeScan prototype it was decided that it was essential to maintain flexibility.

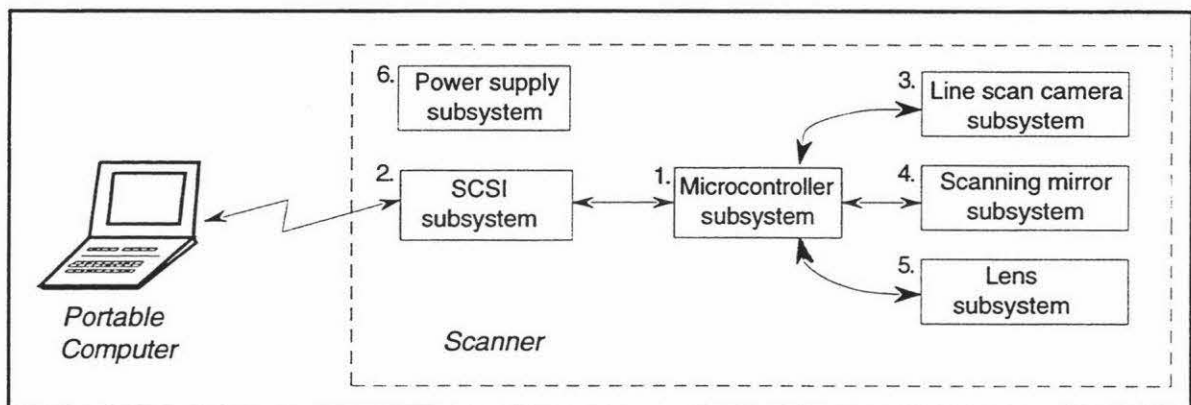


Figure 4.2 - TreeScan scanner functional block diagram

The TreeScan scanner can be divided into six functional blocks (see figure 4.2), the physical layout of which is shown figure 4.3. These functional blocks are:

- | | |
|-------------------------------|------------------------------|
| 1. Microcontroller subsystem | 4. Scanning mirror subsystem |
| 2. SCSI subsystem | 5. Lens subsystem |
| 3. Line scan camera subsystem | 6. Power supply subsystem |

Microcontroller subsystem - Central to the scanner is the microcontroller subsystem. The microcontroller coordinates all functions of the TreeScan scanner and carries out tasks based on the SCC's passed from the portable computer. The microcontroller handles the actual image acquisition and image transfer to the computer, as well as the generation of signals to control the other five functional blocks.

SCSI subsystem - Scanner control commands are passed from the computer to the scanner via a SCSI interface (Small Computer Systems Interface - see section 4.4). The

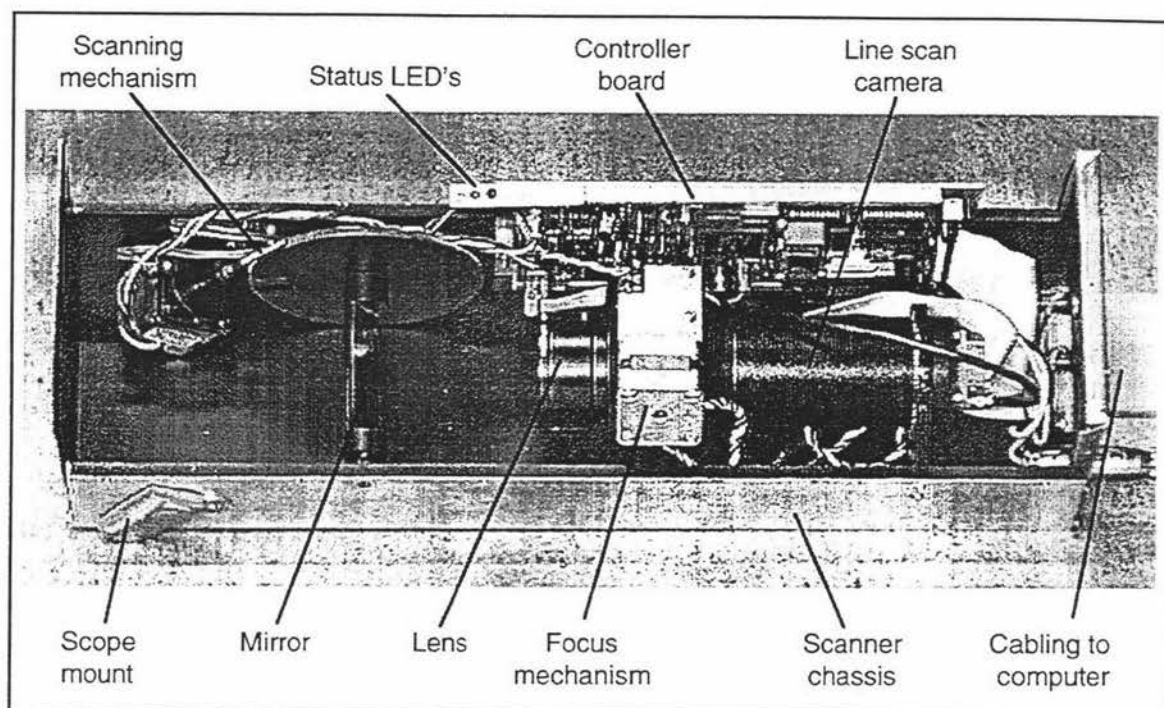


Figure 4.3 - The scanner internal layout

SCSI interface is a high speed communications interface often used to communicate between a computer and peripheral devices. The SCSI specification incorporates a communications protocol that consists of a sequence of bus phases with a complex sequence of control signalling. A SCSI bus controller deals with the bus phases and bus signalling, providing a straight-forward interface to the SCSI bus for the microcontroller.

Line scan camera subsystem - The third important block is the line scan camera subsystem. The image data is captured by the line scan camera and presented to the microcontroller as a series of analog video signals. The microcontroller converts these analog signals to a digital representation which is then sent to the computer using the SCSI controller. The line scan camera captures grey scale image information at a resolution of 1024 pixels per line. The line scan camera is controlled by two timing signals generated by the microcontroller; a line/integration clock and a pixel clock.

Scanning mirror subsystem - The scanning mirror subsystem consists of a mirror mounted on a precision rotation mechanism. As the mechanism is slowly rotated the image is built up one line at a time. The mechanism is rotated using a worm wheel drive shaft attached to a stepper motor. The microcontroller software determines when the stepper motor is rotated. The rotation mechanism is precision machined and mounted on miniature roller bearings. Two optical position sensors are used to detect an exact home position for the mechanism.

Lens subsystem - The fifth functional block is the lens subsystem. The purpose of the lens is to focus an image of a real world object on to the sensor of the line scan camera. The lens has motorised focus control, motorised zoom control and electromechanical aperture control. This allows maximum flexibility during image capture. The infinity position of the focus stepper motor is detected by a limit switch.

Power supply subsystem - The power supply subsystem provides power at the required voltage to all of the above modules from two external batteries. The power supply has two states controlled by the microcontroller:

- **Power save** - during which power is turned off to all of the high consumption components. Only the CMOS microcontroller and SCSI controller are left powered so the computer can still communicate with the scanner.
- **Power on** - during which power is turned on to all components. The scanner is in this state only during image capture.

Figure 4.4 is a signal flow diagram that provides logical details of the signals that pass between individual functional blocks, each of which is discussed in detail in sections 4.3 to 4.6.

The TreeScan system is still undergoing continual improvement. During field trials with the Mk1 prototype several problems were successfully identified (see section 6.2 for a discussion on these). As a result two aspects of the system were redesigned for the Mk2 prototype. The system is currently in the Mk2 prototype stage. Unless specifically stated otherwise, discussions on hardware will apply to both the Mk1 and Mk2 versions of the TreeScan system.

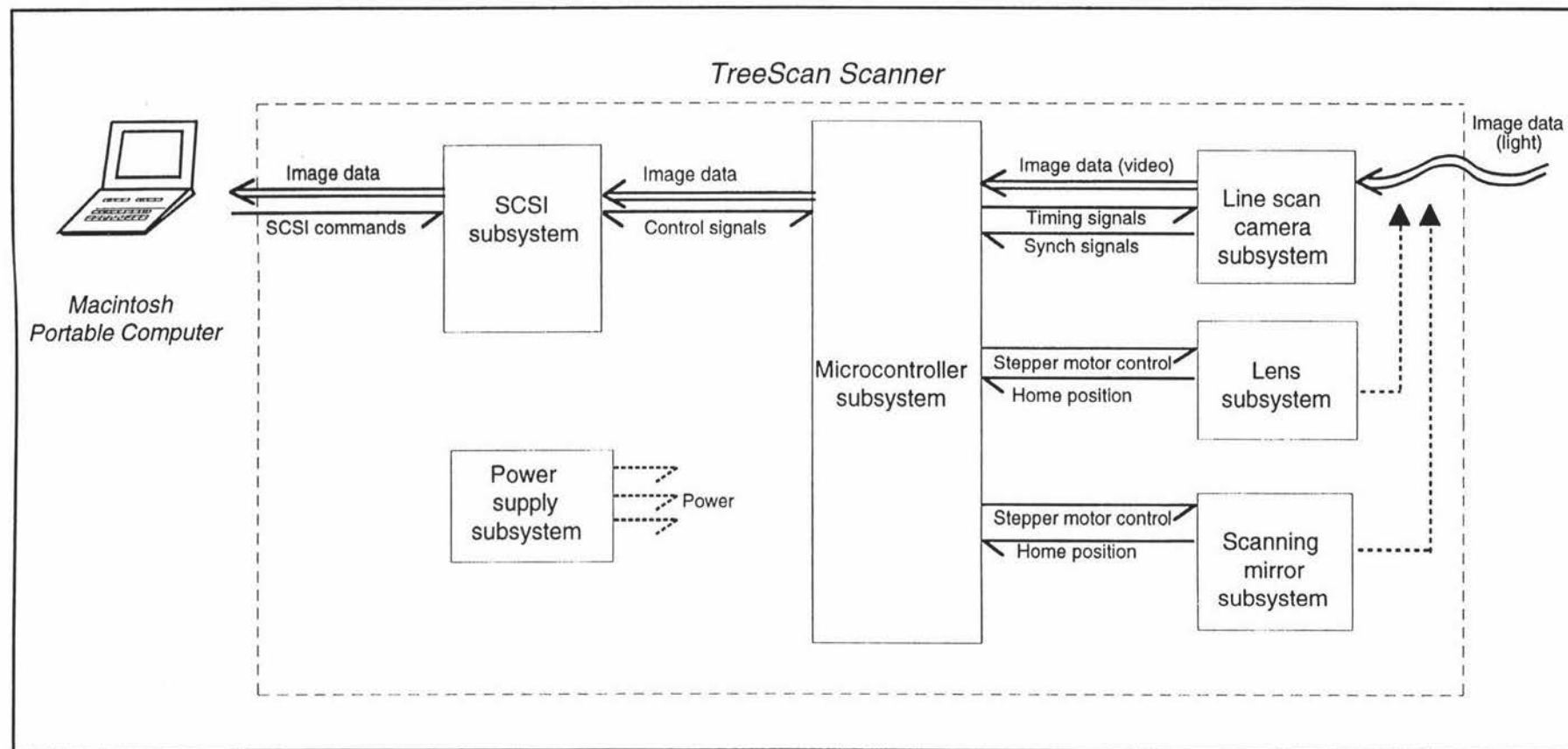


Figure 4.4 - System signal flow diagram

4.2.1 Scanner Controller Board

A printed circuit board (PCB) has been designed and manufactured to accommodate all the scanner electronics. This consists of the microcontroller subsystem self-contained on the PCB, and the driving and interfacing electronics for the other functional blocks.

The PCB is separated into three physical sections with separated power supply sections to reduce possible noise problems:

- **Digital** - Microcontroller and SCSI controller
These are operating at a clock speed of 16 MHz and could cause high frequency noise in the analog sections.
- **Analog** - Line scan camera drivers, stepper motor drivers, and lens drivers
Components in this section draw large currents which could cause supply voltage fluctuations.
- **Analog reference** - A/D reference voltages and video shield
These are isolated from the digital and analog sections to reduce noise on the video signal.

A small plugin daughter board contains the additional lens driving circuitry required for the Mk2 version of the scanner.

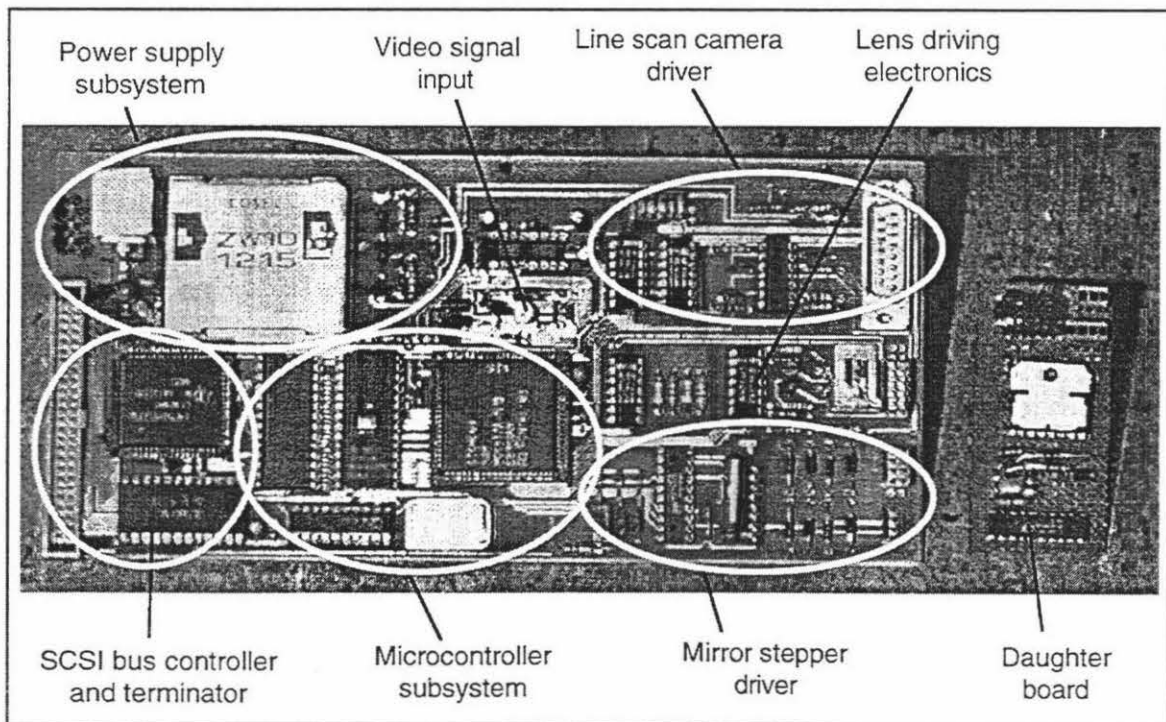


Figure 4.5 - Scanner controller board layout

4.3 Microcontroller Subsystem

General description

At the core of the scanner lies the microcontroller coordinating the operations inside the scanner, many of which are time critical so a dedicated microcontroller is necessary. The microcontroller carries out tasks based on scanner control commands sent from the portable computer. The main task of the microcontroller is to perform the A/D conversion of the analog video data and transfer this image data to the computer. In addition to this the microcontroller must generate signals to control the line scan camera, the lens, the SCSI bus controller, the power supply, and operate to the stepper motors.

The microcontroller used in the scanner is the Siemens 80C517A. The 80C517A is an 8-bit CMOS single chip microcontroller designed for control in hostile environments such as general instrumentation, and industrial and automotive control systems. The 80C517A is a high end member of the 8052 microcontroller family, chosen for its 100 kHz on board A/D converter, the fastest available in this class of microcontroller. The 80C517A has numerous other advanced features including serial communications support, five 8-bit input / output communications ports, four clocks, and a four priority level interrupt handling system.

The microcontroller performs a 10 bit A/D conversion of the video signal, however normally only the top 8 bits are used in the TreeScan system to increase image capture speed. The reason for this is that the result of this A/D conversion is stored in two bytes. The top 8 bits in one byte and bottom 2 bits in another byte. This has the implication that the top 8 bits can be rapidly read out as an 8 bit conversion over the full A/D dynamic range. If however an 8 bit conversion over the lower portion of the A/D dynamic range or a 10 bit conversion with 8 bit lookup table is required the relevant bits of the two bytes need to be combined. This requires extra processing time so increases the time required per pixel conversion. This matter is further discussed in the section on timing in the software chapter (see section 5.2.3.3).

Both memory mapped I/O and port based I/O are used to communicate with other components in the scanner. The interface to the SCSI bus controller is through memory mapped I/O (see memory map - figure 4.7). Communications to all other functional blocks is port based.

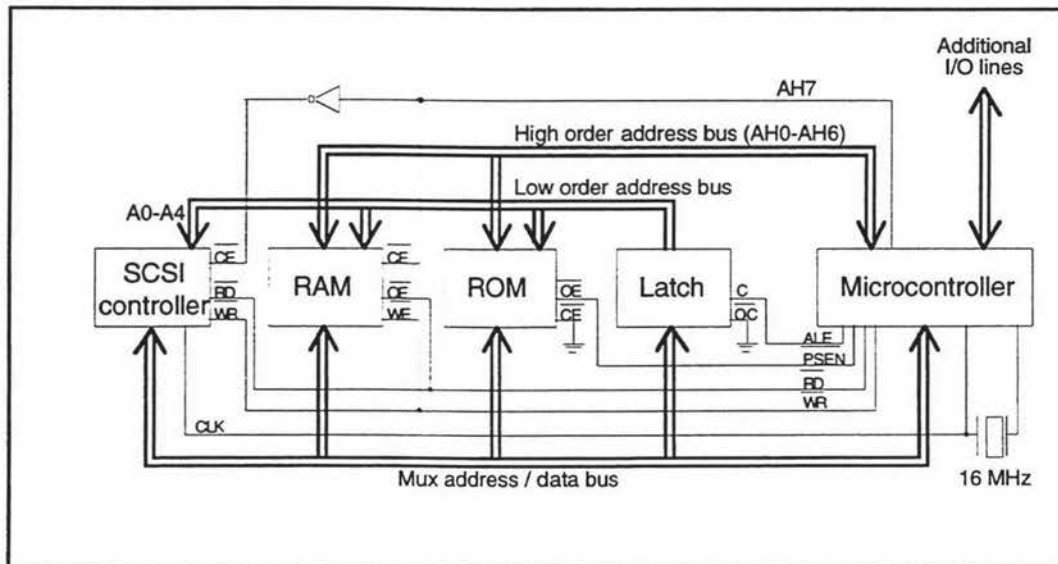


Figure 4.6 - Microcontroller block diagram schematic

Technical description

Associated with the microcontroller is 32 kBytes of EPROM, 32 kBytes of external RAM, an address latch, and an oscillator. The microcontroller code is stored in EPROM and currently takes approximately 10 KBytes. The A/D lookup tables are also stored in EPROM and take up 5 KBytes (5 x 1 KByte). This provides adequate space for further program expansion if required. The external RAM was provided as a precaution in case temporary data storage was required to buffer the image data. This is currently being used to allow lines to be resent if errors occur in the transfer of data over the SCSI interface.

The microcontroller has two ports which are used as buses. A high order address bus and a multiplexed low order address and data bus. A 74HC573 address latch is used to decode the low order address bus from the multiplexed address data bus. The address latch is triggered by the ALE signal. Using the top address line (AH7) either the external RAM or the SCSI controller can be selected and no ^{further} address decoding circuitry is required. The PSEN', RD', WR', and AH7 signals are used to select ROM, RAM, and the SCSI bus controller

The 80C517A will operate on a clock frequency anywhere from 3.5 MHz to 18 MHz. An operating frequency of 16 MHz was chosen because this provides the fastest A/D conversion time of 7 μ s (at 18 MHz A/D conversion time is 12.4 μ s). This clock signal is generated using a 16 MHz crystal oscillator which also provides a clock signal for the SCSI controller.

4.3.1 Microcontroller Subsystem Memory Organisation

The organisation of the memory space of the 80C517 CPU is complicated; detailed information is provided in appendix H. The 80C517 CPU has separate address spaces for program and data memory, and manipulates operands in the four address spaces:

- Up to 64 kBytes of program memory
- Up to 64 kBytes of external data memory
- 256 bytes of internal data memory
- 128 bytes of special function registers

Program memory can either be an external EPROM or up to 32 kBytes of factory programmed ROM on the micro controller chip. The active program memory is determined by the state of the EA pin during powerup.

There are two forms of **external data memory**; up to 64 kByte external RAM and 2 kBytes of on chip XRAM. The XRAM is accessed using identical instructions to those used for accessing external RAM but with bit 1 of the SYSCON register set.

All registers, except the program counter and four general purpose register banks, reside in the **special function register** (SFR) area. The SFR's include arithmetic registers, pointers and registers to provide an interface between on chip peripherals. Registers which lie on 8 byte boundaries are bit addressable.

The **internal RAM** contains four banks of registers and a 128 bit bit-addressable section overlapping a part of the internal RAM. The stack pointer is initialised to 08h in internal RAM on reset. There is an address overlap between the upper 128 bytes of

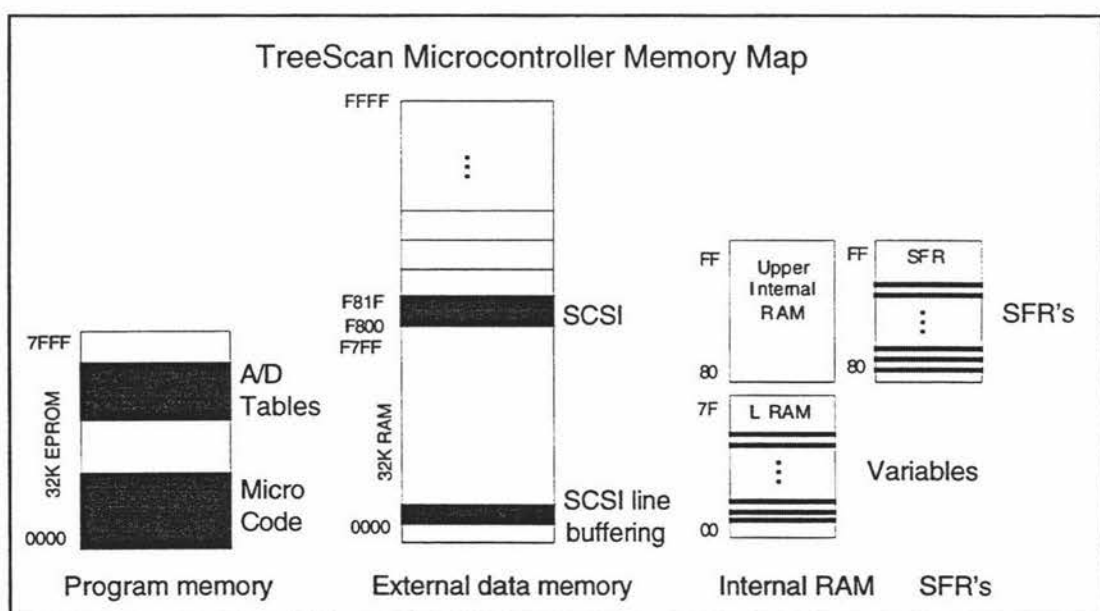


Figure 4.7 - Microcontroller memory map

internal RAM and the SFR's. The addressing mode used determines whether the SFRs are addressed or whether internal RAM is addressed.

The TreeScan scanner microcontroller uses the following sections of the 80C517A memory space (see figure 4.7) :

- 32 kByte EPROM to store the microcontroller code and A/D lookup tables.
- 32 kByte RAM of which 1 kByte is used to buffer the SCSI transfer.
- SCSI controller registers repeatedly mapped into the top 32 kBytes of external data memory.
- The lower internal RAM for working variables.
- The special function registers.

4.3.2 Microcontroller Subsystem Memory Timing

In microprocessor design any external device must be fast enough to match the microprocessor read and write cycle timing. Problems are often encountered with EPROM read cycles. In this case:

- Upon the falling edge of ALE the address latch is triggered. Assuming a maximum delay of 25 nS within the latch the low order address bus data becomes valid no later than 25 nS after falling edge of ALE. The data bus is expected to be valid no later than 233 nS after the falling edge of ALE. This allows the EPROM 208 nS address access time from valid addressing to valid data (see figure 4.8).
- PSEN' is used as EPROM output enable. PSEN' is asserted a minimum of 150 nS before valid data, this allows for an OE' to output delay of 150 nS.

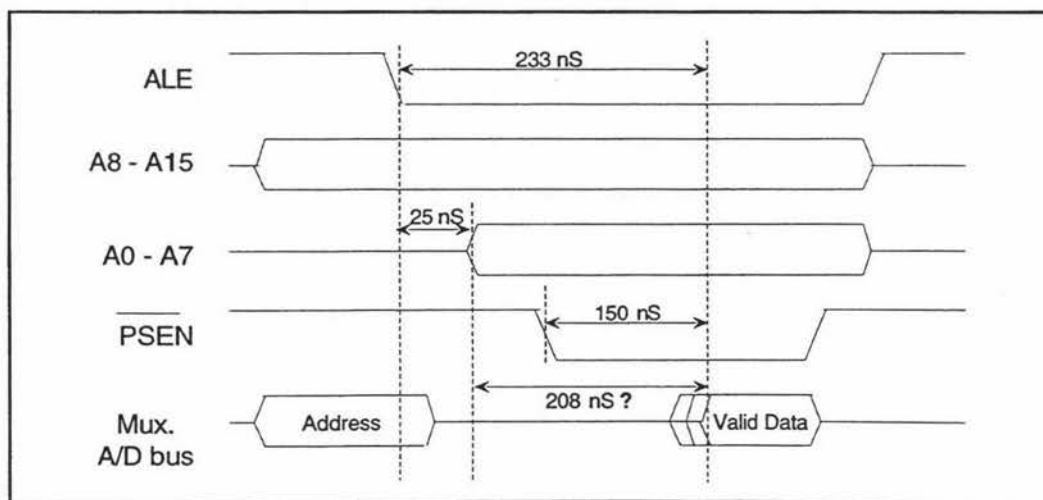


Figure 4.8 - EPROM read cycle timing

EPROMs must be selected which fit the above criteria. 200 nS EPROMs (75 nS OE' to output delay) would be usable but barely under the 208 nS address access time. To be safe 170 nS EPROMs were used (70 nS OE' to output delay).

4.4 SCSI Subsystem

General description

The SCSI subsystem provides a two way communications interface between the microcontroller and the Macintosh computer over a SCSI communications bus using a SCSI bus controller. This interface is used to send scanner control commands to the scanner and transfer the image data back to the computer.

SCSI stands for *Small Computer Systems Interface* and is a high speed, flexible communications interface commonly used to connect peripheral devices to computers (most hard disks are SCSI). The SCSI interface allows for multiple (up to seven) devices to be attached to a single SCSI bus using logical addressing, and allows for data rates up to 4 Mbytes per second. The SCSI communications protocol consists of a sequence of bus phases mediated by a complex sequence of control signals (see section 4.4.1 for a discussion on implementing SCSI).

A SCSI bus controller (SBC) chip provides a simple interface to the SCSI bus for a microcontroller. At the time of development, SCSI bus controllers were available from Texas Instruments, Western Digital, and AMD. The SN75C091A SCSI controller from Texas Instruments was incorporated into the design based on a short chip delivery time and the availability of reference information.

Several problems were encountered while implementing the SCSI subsystem. The main obstacle was a result of unexpected timing fluctuations. Noise problems were also experienced on the development circuit implemented on veroboard. These are further discussed in section 4.4.3.

Manufacturer	SCSI Chip	Availability
Texas Instruments	SN75C091A	Available
Western Digital	WD33C93B	Available - Delayed reference data
AMD	AM53C80APC	Available - Delayed reference data
Hitachi	64951	Not Available

Table 4.1 - Availability of SCSI bus controllers

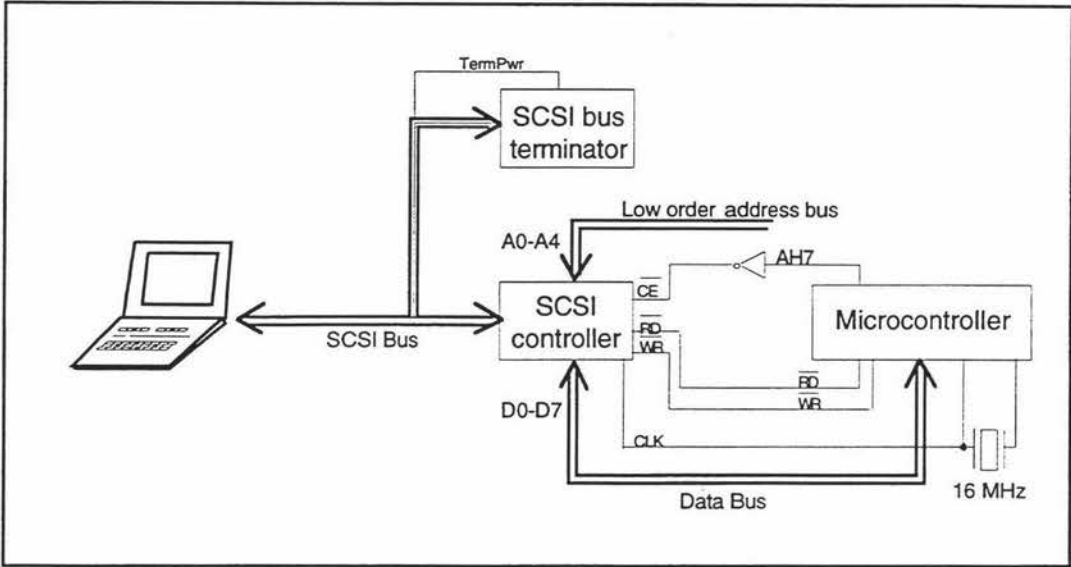


Figure 4.9 - SCSI block diagram schematic

Technical description

The SCSI subsystem consists of a SCSI bus controller and a SCSI bus terminator inside the scanner. Both the terminator and bus controller are attached to the SCSI bus which is connected to the computer.

The SN75C091A SCSI bus controller consists of a 68 pin square PLCC package. The microcontroller communicates with the SCSI controller using 32 bytes of memory mapped registers. The SCSI controller is selected by the inverted top address line. By using the top address line no address decoding circuitry is required. This maps the SCSI registers repeatedly into the top 32K of the microcontroller address space.

The SCSI bus must be correctly terminated. The scanner is internally terminated so it can only be the final device on a SCSI chain. The terminator used in the scanner is a UC5601 chip and powered from the TERMPower line on the SCSI bus.

4.4.1 Implementing SCSI : Design Specifications

The SCSI interface is defined by the ANSI X3.131 - 1986 standard (ANSI, 1986). The standard defines the mechanical, electrical, and functional requirements for the SCSI bus, and the protocol command sets. The SCSI bus consists of 18 signal lines, nine of which are control signal lines and nine of which are data signal lines. The SCSI communications protocol consists of a sequence of bus phases mediated by a complex sequence of control signals.

The SCSI interface allows for multiple devices attached to a single SCSI bus using logical addressing. The SCSI bus consists of a series of daisy-chained devices terminated at each end. The specification allows for data rates up to 4M bytes per second and cable lengths up to 25 m dependent on the circuit implementation. Transfers may be implemented using either a synchronous or an asynchronous protocol.

The standard specifies a maximum cable length of 6 m for an implementation with single ended drivers and receivers. A single ended implementation should use a cable with a 132 ohm characteristic impedance to match the terminators, and minimal media discontinuities to reduce signal reflections. Ideal conditions are not usually attainable and an implementation may require the trade-offs in shielding effectiveness, cable length, the number of loads, transfer rates, and cost to achieve satisfactory system operation (ANSI, 1986). Practically, this implies that a maximum cable length of 1.8m (6 ft) is used to prevent data corruption during transfer. A cable length of 1.8 m is just adequate for the TreeScan system.

A brief description of the SCSI bus protocols and general SCSI commands is provided here. Further details are presented in appendix I, or can be found in the reference material (ANSI, 1986) .

4.4.1.1 SCSI Bus Protocols

The SCSI protocol contains eight distinct **phases** (see next page). The SCSI bus can only be in one of these eight phases at any one time. A SCSI operation is a completed SCSI command or data transfer. A single SCSI operation consists of the execution of a carefully controlled sequence of these bus phases.

The device that requests a SCSI operation is called the **initiator**. The device that performs the operation requested by the initiator is the **target**. During a SCSI operation control of the bus is handed back and forth between the initiator and the target until the operation is complete. Only a device that is in control of the SCSI bus may change the bus phase.

The eight bus phases consist of the following (see appendix I):

- BUS FREE Phase
- ARBITRATION Phase
- SELECTION Phase
- RESELECTION Phase
- COMMAND Phase \
- DATA Phase | Collectively called the
- STATUS Phase | INFORMATION Phase
- MESSAGE Phase /

A completed SCSI operation will start with a BUS FREE phase and must proceed through an ARBITRATION phase, SELECTION phase, COMMAND phase, STATUS phase, and a MESSAGE phase. In addition to this the SCSI operation may include a RESELECTION phase and a DATA phase. This sequence can only be broken through a timeout or the undesirable assertion of the bus RESET signal at which time the bus must be released to the BUS FREE phase.

During these bus phases the bus control signals are asserted in a complicated control and handshaking sequence. The sequence the control signals may be asserted is specified in the ANSI standard. A typical SCSI transfer is discussed in section 5.2.4. Minimal and maximal duration between signal transitions is also specified in the standard. The SCSI bus signals are listed below (all signals are active low):

- **Control signals:** BSY, MSG, SEL, REQ, C/D, ACK, I/O, ATN, RST
- **Data signals:** SD0 - SD7, SDP

4.4.1.2 General SCSI Commands

At a higher level, SCSI commands are sent from the computer to the microcontroller. This consists of the transfer of a command descriptor block. A command descriptor block is a data structure containing a command opcode and parameters associated with this opcode. The command descriptor block may be six, ten, or twelve bytes long.

The first byte of the command descriptor block contains the **operation code**. The operation code is the SCSI command number. The top three bits of an operation code specify the group code. SCSI commands fall in several categories based on this group code:

- | | | | |
|--------------------|---------------------|--------------------|--------------------|
| Group 0 | : six byte commands | Group 5 | : 12 byte commands |
| Group 1 | : ten byte commands | Group 6 - 7 | : Vendor unique |
| Group 2 - 4 | : Reserved | | |

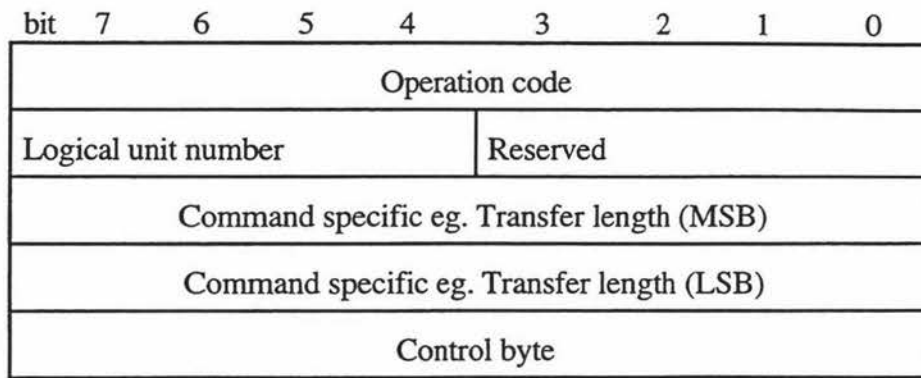


Figure 4.10 - Typical command descriptor block (ANSI, 1986)

A typical command transfer block would contain the information shown in figure 4.10.

In order for a device to adhere to the SCSI specification a number of general commands must be implemented. Out of 256 available commands four commands are classed as mandatory, four commands are for devices that support independent self configuring software, twenty two commands are optional, twenty three commands are vendor specific, with the rest reserved for future use.

The classification of commands as mandatory or optional is dependent on the device type. Device types include direct access devices, sequential access devices, printer devices, processor devices and WORM devices (see appendix I).

The TreeScan system implements 20 commands, all within the six byte command range.

4.4.2 SCSI Bus Controller (SN75C091A)

The SN75C091A SBC manufactured by Texas Instruments is a single ended flexible SCSI implementation for microprocessors. It provides DMA or programmed I/O capabilities and can be interrupt driven to minimise host polling. The SN75C091A can execute multiphase commands to minimise host interrupts. Chip access is provided through 32 directly addressable registers (Texas Instruments, 1990).

The SBC is driven by chip commands written to the COMMAND register. These commands are instructions from the microcontroller to the SBC to modify the current bus phase or transfer data. These commands fall in three categories:

- Non interrupting commands
- Single phase interrupting commands
- Multi phase interrupting commands

The SBC is controlled by the background loop of the microcontroller software. The TreeScan system uses mainly single phase interrupting commands in a processor polled loop (see section 5.2.4). Appendix J provides more detailed SBC specifications.

4.4.3 SCSI Subsystem Development Obstacles

Several problems were encountered during the development of the SCSI interface:

Problems were experienced with the TreeScan development boards;

1. Noise problems were experienced during data transfer over the SCSI bus connection to the development board. This was probably due to lack of shielding of the SCSI bus connection to the SCSI bus controller (developed on the veroboard). This intermittent problem was solved by using the custom designed printed circuit board.
2. For testing the SBC was piggybacked in place of another memory mapped I/O device on the microcontroller development board. The trigger pulse of the replaced I/O device was insufficient in duration to latch the SCSI registers.

The second major problem involved unexpected timing fluctuations of the SCSI interface and was discovered during the development of the *SCSI transfer algorithm* (see section 5.2.4).

1. Although SCSI is a handshake system, with a 'wait if not ready' flag, there is insufficient time in the main capture loop to check this flag. It is assumed the SCSI interface (4 MBytes/s) is able to keep up with A/D conversion (100 kHz). At times the SCSI controller of the Macintosh was unable to receive data for intervals of approx 15 mS in duration. This was probably due to background operating system tasks and caused image bytes to be lost as the reading out of the line scan camera data could not be delayed. An error detection and line resend scheme has been implemented (see section 5.2.4 for further detail).
2. The SCSI interface was slow to react to SCSI phase changes introducing a minimum SCSI transfer duration of approximately 170 mS (see section 5.2.4 for further detail). This transfer duration should be in the order of nanoseconds, and its cause should be further investigated.
3. During the tracking of the above problems the SCSI clock frequency was increased to 20 MHz. The microcontroller still operated at 16 MHz. This introduced a timing mismatch which caused occasional (approx. 1 byte in 50 000) bytes to be gained during the transfer. This was rectified as soon as it was discovered, but the consequence of this was that the error detection and resend scheme was complicated unnecessarily.

4.5 Line Scan Camera Subsystem

General description

The line scan camera subsystem captures image data and converts it to an analog video signal which is used to generate the captured image. The line scan camera subsystem consists of a CCD line scan camera (LSC), interfacing buffers, and the analog reference section of the controller board.

The line scan camera subsystem is a very important section of the TreeScan scanner. The video signal determines the image quality so it is important to ensure the video signal is well shielded and that the microcontroller analog reference voltages are stable. The image is captured at 256 level greyscale (8 bit digitisation) which is well within the 2000:1 RMS dynamic range of the line scan camera.

The line scan camera being used is a Loral Fairchild CAM 1301R camera designed for incorporation into non-contact electro-optical measurement and process control systems. The CAM 1301R has a resolution of 1024 x 1 pixels and incorporates anti-blooming and electronic exposure control. The camera accepts standard C mount lenses, with the option of using bayonet Nikon and Olympus mounts also available.

A number of sources of line scan cameras or line scan camera systems were considered which are summarised in table 4.2. The Leaf and Chinon cameras are complete area scanning systems which make use of a line scan approach similar to the TreeScan system. Modification of these systems was investigated but not pursued.

Line scan cameras are available with analog or digital output; digital output is preferred but the cost of these cameras is very high. Cameras with analog output are less specialised devices and are considerably less expensive. Analog cameras require signal A/D conversion by some external device. This is completed by the microcontroller in the TreeScan system. It was decided to use the Loral Fairchild CAM 1301R line scan camera based on cost and delivery time.

Supplier	Camera type	Interface	Resolution	Price
Loral Fairchild	CAM/CCD 1000 series	Analog	512 to 6000	Medium
DALSA Inc	CL-CX series	Analog, digital	128 to 4096	High
i2S	iDC / IVC 100 series	Analog, digital	256 to 3456	High
Pulnix	J series miniature LSC	Analog	1024 to 5000	Medium
Leaf - <i>System</i>	Leaf digital camera	SCSI	2000 x 1500	Very high
Chinon - <i>System</i>	DS-3000 scanner	Digital, SCSI	3328 x 2300	Low

Table 4.2 - Line scan cameras available

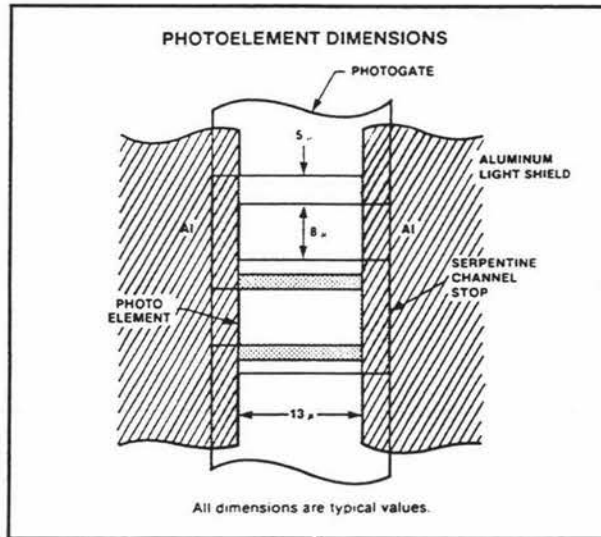


Figure 4.11 - Imaging sensor photosite layout

The CAM 1301R is controlled by two timing signals. One signal that controls the line rate and integration time, and another signal that clocks the video data out. The camera derives two synchronisation signals from these timing signals which are returned and used to synchronise the microcontroller A/D conversion (see section 4.5.2).

The CAM 1301R contains a Loral Fairchild CCD134 imaging sensor. The CCD134 sensor contains 1062 array elements of which 1024 are photosensitive. Each photosite is $13\ \mu\text{m} \times 13\ \mu\text{m}$ on a centre spacing of $13\ \mu\text{m}$ (see figure 4.11). Between photosites there is a $5\ \mu\text{m}$ serpentine stop channel providing an active area per pixel of $13 \times 8\ \mu\text{m}$. The sensor has a length of 13.8 mm. This relates to a one inch format for lens requirements.

The main obstacle encountered during the development of the line scan camera subsystem was the timing constraints the line scan camera imposes on the rest of the TreeScan system. Image data is being clocked out of the camera at approximately 100 kHz, during which time the subsequent line is being exposed. This implies the data being clocked out cannot be slowed or temporarily halted as this invalidates the data of the subsequent image line. The implications of this are further discussed in the software chapter, see section 5.2.1.

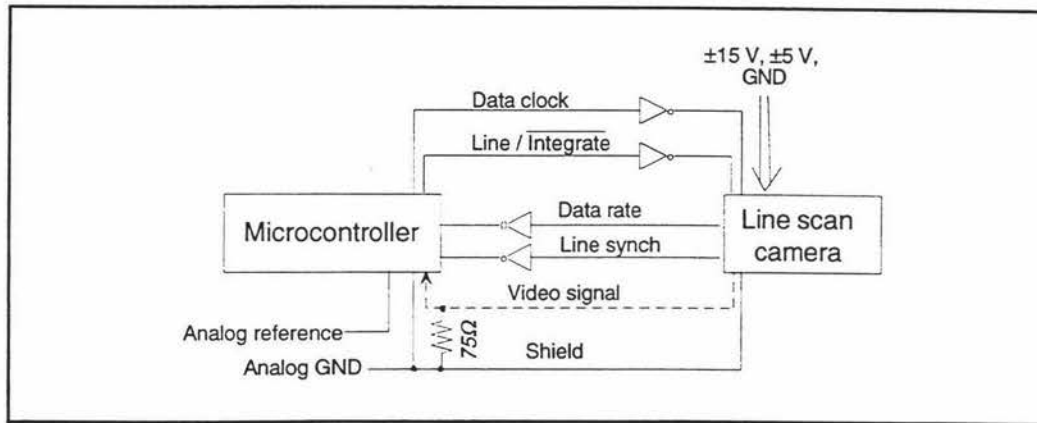


Figure 4.12 - LSC interface block diagram schematic

Technical description

The line scan camera subsystem consists of a line scan camera, interfacing buffers, and analog reference section of the controller board.

The CAM 1301R can be operated at a wide range of frequencies (up to 20 MHz) and is controlled by two timing signals generated as pulse width modulated signals by the microcontroller software, the DATA CLOCK and a LINE/INTEGRATE clock - see section 4.5.2. These camera timing inputs are differential with signal levels converted to TTL levels by internal differential line receivers. At low data rates (<1 MHz) and short cable lengths (<6 ft) single ended TTL input clock signals may be used provided the negative differential input is biased at +1V. This is the camera connection scheme used in the TreeScan scanner. All the clock signals sent to and from the camera are buffered by 74HC04 inverters to act as line drivers and receivers.

The video output is available as a 75 Ω source impedance signal on either of two coaxial connectors on the rear of the camera. The video signal has a peak of +1 volt at sensor saturation.

The video signal is terminated by a 75 Ω resistor and passed to the microcontroller for A/D conversion. The microcontroller has an analog GND and +1 volt reference for the A/D convertor which are isolated from the rest of the circuit using ferrite beads for noise suppression.

The camera requires power supply inputs of +5, +15 and -15 volts DC. Internal regulators and filters provide noise immunity for the CCD sensor bias voltages.

4.5.1 Imaging Sensor Spectral Response

The spectral response of the CCD134 imaging sensor covers light over the wavelengths of 35 nm to 1000 nm with a peak responsivity of $5.8 \text{ V}\mu\text{J}^{-1}\text{cm}^{-2}$ at 800 nm.

A Schott KG-1 infra-red cut off filter is made part of the standard camera. The filter transmission convolved with the spectral response of the imaging sensor gives the camera a response from about 350 nm to 800 nm, with a peak response at 600 nm (Loral Fairchild, 1991). This spectral response covers most of the visible spectrum with the greatest response in the red colour band.

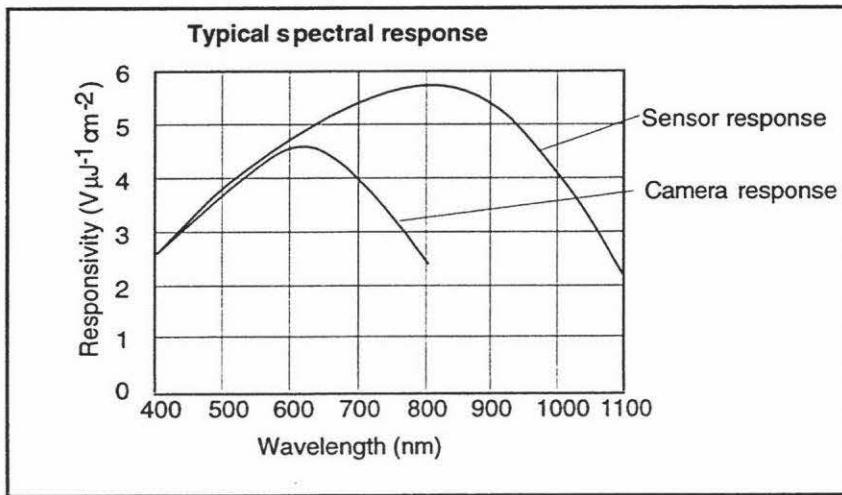


Figure 4.13 - CCD sensor spectral response

4.5.2 Line Scan Camera Subsystem Signal Timing

The line scan camera is controlled by two timing signals; a **DATA CLOCK** and a **LINE/INTEGRATE** clock. The microcontroller generates both these timing signals in software using on board timers.

- The **DATA CLOCK** rate determines the rate at which video data is clocked out of the camera. The **DATA CLOCK** rate is set at approximately 100 kHz, well under the maximum camera clock rate of 20MHz.
- The **LINE/INTEGRATE** clock determines both the line scan rate and the optical integration time (or exposure time). Integration time is controlled by the duty cycle of **LINE/INTEGRATE** clock, while the line scan rate is controlled by its frequency (see figure 4.14).

Note : Terms in capital letters refer to actual signal lines.

The TreeScan system must operate under low light forestry conditions. It has been empirically determined that the integration time will typically be somewhere between 2 and 50 mS using a large aperture lens. The frequency of the LINE/INTEGRATE clock varies with integration time, with a maximum frequency of 50 Hz (scan rate of 50 lines per second) determined by the A/D speed of the microcontroller .

The above timing signals are returned from the line scan camera as synchronisation signals:

- The **DATA RATE** signal is the **DATA CLOCK** delayed by approximately 10 nS.
- The **LINE SYNC** signal indicates the start of line time and is derived from the **LINE/INTEGRATE** clock with a delay of 1 - 2 **DATA CLOCK** cycles.

The **DATA RATE** signal is used to trigger the A/D conversion of each pixel. The **LINE SYNC** signal is used for line synchronisation and precedes active video by 24 **DATA CLOCK** cycles.

The video signal contains 1024 individual pixel levels which corresponds to 1024 **DATA CLOCK** cycles. Preceding and following the active video there is a 3 clock cycle dark reference. Following the final dark reference there is a two clock cycle white reference. As a result, a minimum of 1062 **DATA CLOCK** cycles are required to fully clock out the CCD134 sensor.

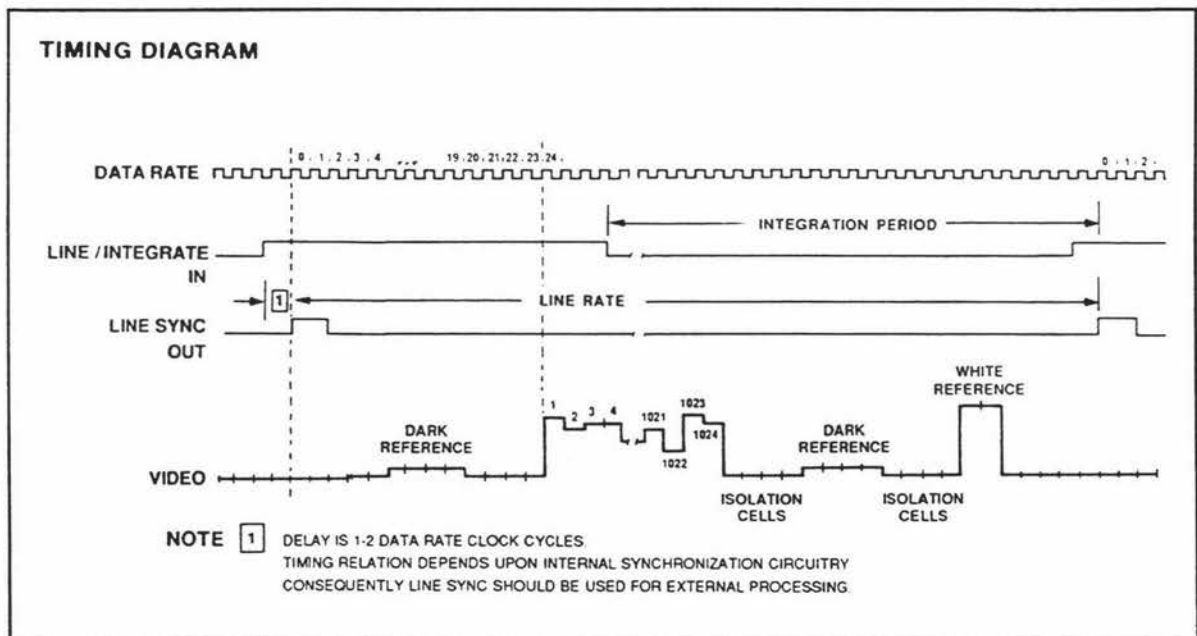


Figure 4.14 - Line scan camera timing

(LSC pamphlet - Loral Fairchild, 1992)

4.6 Additional Hardware

This section describes additional TreeScan hardware. This includes a discussion on the final three functional blocks (scanning mirror subsystem, lens subsystem, and power supply subsystem) and a description of the scanner chassis, the protective carrying cases, and the hardware used to monitor scanner status.

4.6.1 Scanning Mirror Subsystem

General description

Images are built up one line at a time by incrementally rotating the scanning mirror. The scanning mirror subsystem, which provides control over the positioning of this mirror, consists of a mirror mounted on a precision rotation mechanism that is driven by a stepper motor.

The custom made precision rotation mechanism consists of a shaft upon which the mirror is mounted and a wormwheel drive. The wormwheel drive has a direct 728 to 1 reduction ratio and has been engineered to ensure minimal backlash in the gears. The shaft is mounted on ball bearings for smooth rotation. The reduction and precision of the rotation mechanism along with the step size of the stepper motor determines the vertical pixel spacing. The design and construction of the precision mechanism was completed by Mr Thomas Look.

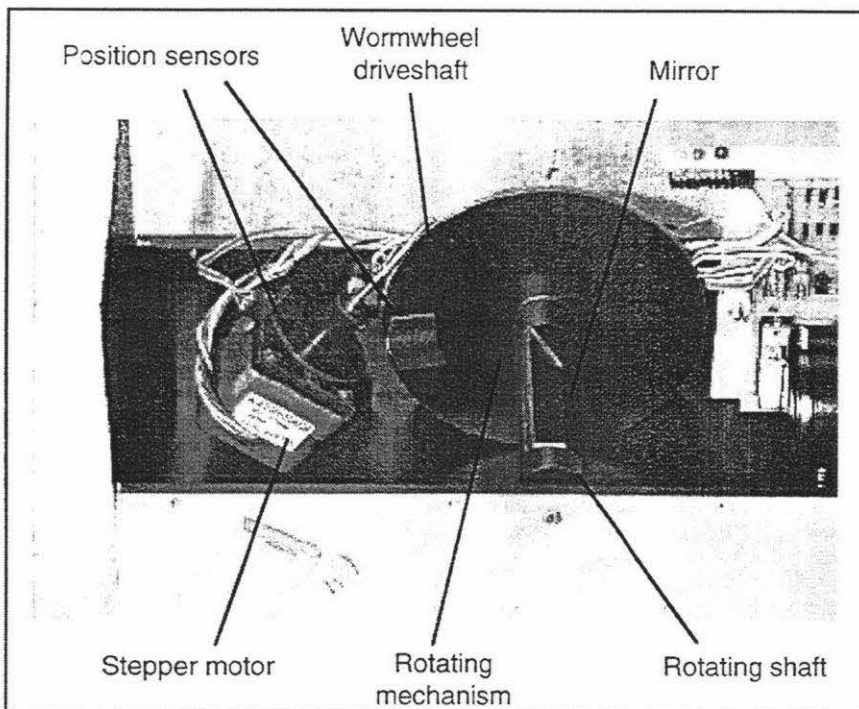


Figure 4.15 - Scanning mirror assembly

Two optical position sensors are used to determine the exact home position of the mechanism. The first sensor determines the position of the large wheel, with the second sensor detecting the position of the stepper motor within a single revolution. These are directly interfaced to the microcontroller.

A four phase, bipolar stepper motor is used to drive the precision rotation mechanism. The stepper motor has a step angle of 1.8 degrees per step which relates to 200 steps per revolution of the stepper motor. This provides the flexibility of double the required resolution with 16000 stepper steps over the full tree height.

Stepper motor detent torque of 70 mNm is more than adequate. The main shaft rotates freely and with a geared reduction of 728 : 1 results in negligible stepper motor torque requirements to rotate the mechanism. This allows the stepper motor to operate near its maximum rate of approximately 400 steps per second for this supply voltage.

Technical description

This section describes both the mechanics of the rotation mechanism as well as the stepper motor control circuitry.

Mechanical design

The mirror used to reflect the image to be captured is a float glass mirror with a metallic enhanced front surface coating suitable for wavelengths of 400 to 750 nm (>90% reflectance of all wavelengths). The glass thickness is 3.05 mm.

The wormwheel drive consists of a 5.7 cm brass wheel with an M3 thread machined on the outer edge. A shaft was machined to mesh with this edge and is attached to the stepper motor shaft. The arrangement can be seen in figure 4.15.

The precision rotation mechanism must produce accurate and repeatable mirror positioning with minimal gear backlash. Vertical image pixel spacing of 8000 pixels for a 40 metre tree required pixel angular spacing to be 1/100 of a degree. Thus the mirror rotation per line must be 0.005 degrees. These are fine requirements given the machining facilities available. The shaft alignment had a similar requirement of bearing movement of less than 1 pixel. This means the shaft alignment must be repeatable to within 0.01 degrees (or shaft end position repeatable to ± 0.017 mm).

After machining the repeatability of the mechanism was tested by measuring the repeatability of a reflected laser point on a distant wall. There was no backlash or alignment movement to the limits of measurability ± 0.01 degrees (or ± 2 pixels).

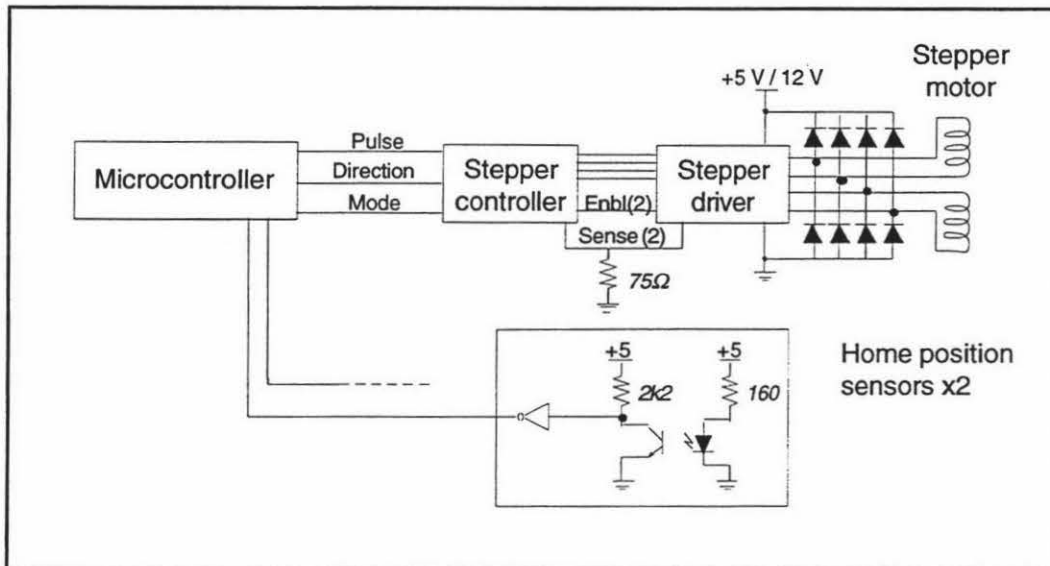


Figure 4.16 - Stepper motor controller block diagram schematic

Stepper motor control

The four phase stepper motor is driven using a two phase bipolar stepper motor control circuit. The circuit consists of a stepper controller IC (L297 manufactured by SGS-Thompson) used in conjunction with a darlington stepper motor driver chip (L298N).

Two modifications were made to the stepper motor of the Mk2 TreeScan unit.

- The Mk1 unit had a 1.8 degree per step stepper motor, with two motor steps for each mirror step (providing double the required resolution). In the Mk2 unit this motor was replaced by a 3.75 degree per step stepper motor. This allowed for a speed increase of 2 ms per line during image capture, and a speed increase by a factor of 2 when homing the mechanism.
- The stepper motor supply voltage was increased from 5 V to +12V. This also allowed for a further speed increase.

Time for 8000 mirror steps (Mk1) = 16000 stepper steps @ 330 steps /s
= 48 seconds

Time for 8000 mirror steps (Mk2) = 8000 stepper steps @ 500 steps /s
= 16 seconds

The Mk2 stepper motor may be driven in full step or half step mode to provide smaller step sizes if required.

4.6.2 Lens Subsystem

General description

The lens focuses an optical image of a real world object on the line scan camera imaging sensor. Variations in object distance and lighting conditions require the focus and aperture to be electronically adjustable by the scanner. Focal length is the third lens parameter which can be modified on demand, provided a zoom lens is used.

The Mk1 system was designed with a Cosmimar TV zoom lens with motorised control of focus and zoom, and electronic aperture control. This provides the microcontroller with full control of the lens. Very good images were captured with this lens.

Unfortunately some problems were highlighted with the use of the Cosmimar lens during the calibration and characterisation of the scanner:

1. The Mk1 lens did not have accurate focus positioning as it used DC servo motors.
2. Lighting in forest conditions was very low so a lens was required that provided more light.
3. The lens was unsuitable for the relatively long CCD imaging sensor. Light falling on the outer edges of the sensor was being attenuated through vignetting.

A different lens was incorporated into the Mk2 system; a manual lens of fixed focal length. A manual lens was used as this provides a larger aperture. A stepper motor was mounted on the focus ring to provide the microcontroller with motorised focus control (see figure 4.17).

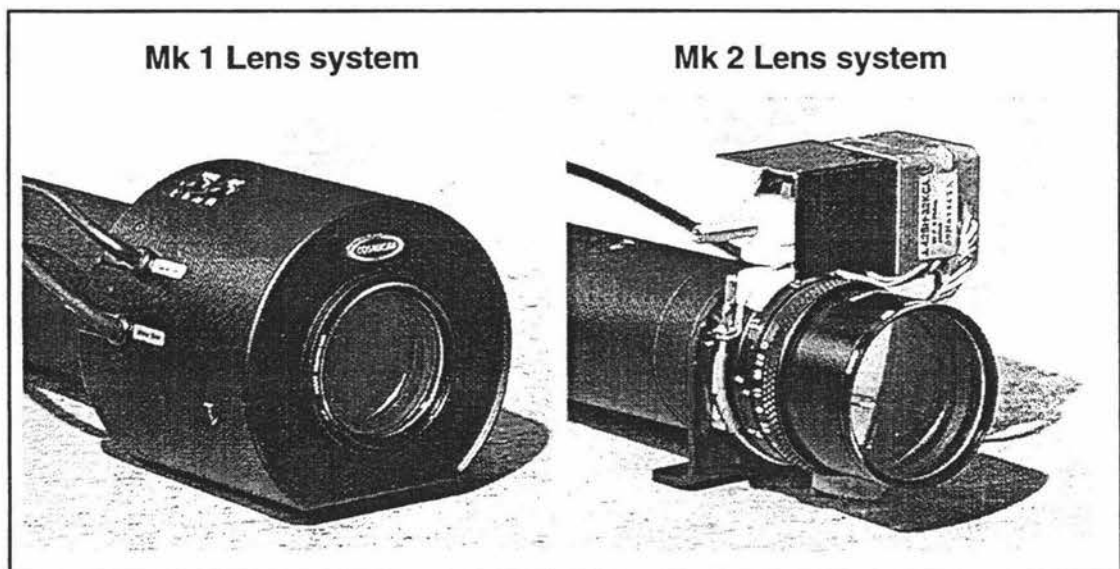


Figure 4.17 - Mk1 and Mk2 lens assembly

Technical description

Mk1 Cosmocar lens

The Mk1 version of the TreeScan scanner contains a Cosmocar C6Z1218M2ESP motorised zoom lens with auto iris. The 2/3 inch format lens has a C mount and a focal length range of 12.5 to 75 mm. The lens has a maximum aperture of f 1.8 with a range of f 1.8 to f 360.

Both zoom and focus are controlled using 12 V DC servo motors. It takes approximately five seconds to drive the servos from one end of their operating range to the other. The direction of the servo motors may be reversed by reversing the polarity on the driving signal. The lens also has an auto iris which adjusts the aperture automatically based on an input video signal.

The Cosmocar lens was purchased because it provided flexibility in the implementation of the TreeScan scanner. The zoom provides flexibility in horizontal image resolution and the auto iris allows automatic aperture adjustment without microcontroller intervention.

The driving circuit for the Mk1 lens provides a variable voltage to modify the operation of the automatic aperture adjustment, and ± 10 V to drive the DC servo motors.

Mk2 lens

The Mk2 version of the TreeScan scanner contains a 75 mm fixed focal length manual lens. This lens, a one inch format TV lens with C mount, has a maximum aperture of f 1.4 with a range of f 1.4 to f 22. It was decided to purchase a manual lens and add motorised controls as this was the only one inch format, large aperture lens readily available.

A stepper motor identical to the mirror drive stepper motor was attached to the focus ring to provide the microcontroller with motorised focus control. An additional stepper driving circuit was built as an extension module and attached using the expansion area on the PCB. As a result a considerable amount of the Mk1 driving electronics became redundant. These driving circuits are still available on the board for backward compatibility.

By using the stepper motor in conjunction with an infinity position sensor absolute focus position information is available for the lens. This provides the capability of making a 'blind focus' during an image capture by estimating where the scanner should be focused, and moving the lens focus to this position.

The problems with the Cosmocar lens have been successfully solved. However images captured with the Mk2 system appear to be somewhat blurred compared to the best

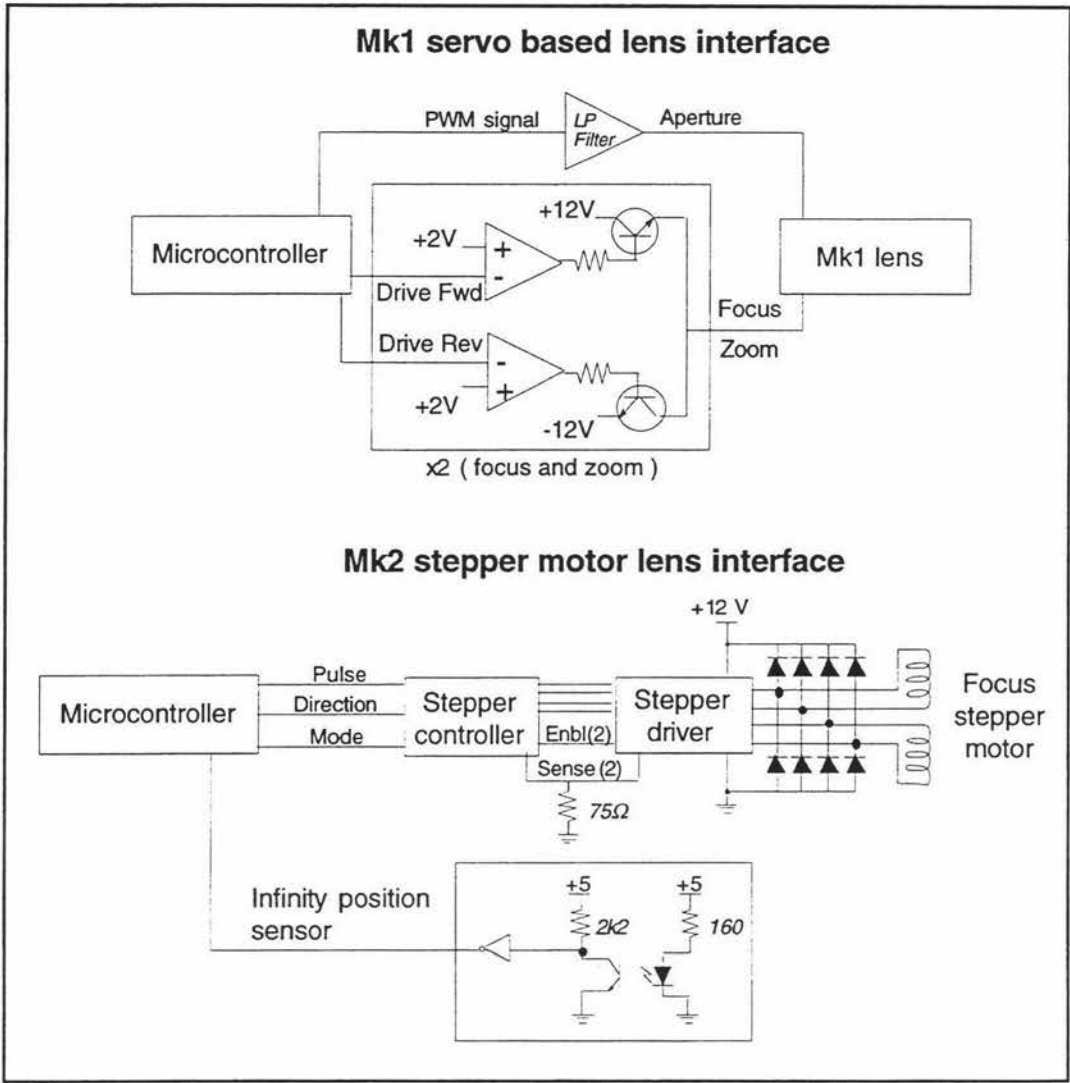


Figure 4.18 - Mk1 and Mk2 lens driving interface

results obtained with the Mk1 system. This could be due to focus problems or lens aberration at large apertures. Information regarding lens spectral response and lens modulation transfer function is not available. This problem is under investigation by Mr Aaron Drysdale as part of his masterate.

4.6.3 Power Supply Subsystem

General description

The scanner is a portable unit that must operate on battery power while in operation. It is important to make the power supply as small and lightweight as possible while retaining the ability to store adequate charge. The power supply of the computer and scanner are separate; the computer runs off its own internal two batteries, and the scanner operates off two sealed 6 volt lead acid batteries.

A variety of voltages are required to power the TreeScan scanner. The power supply takes a nominal 6 V DC and 12 V DC battery input and produces +5V, +10V, -10V, +15V and -15V DC regulated DC supplies.

The power supply has two states :

- **Power save** during which a relay turns off power to all of the high consumption components. Only the CMOS microcontroller and SCSI controller are left powered so the computer can still communicate with the scanner.
- **Power on** during which power is turned on to all high consumption components. The scanner is in this state only during image capture.

To conserve battery power the scanner is in the standby power save state at all times unless actually capturing an image.

Two common types of battery used to power portable instruments are Nicad cells and lead acid batteries. The use of each of these has advantages and disadvantages:

1. **Nicad batteries** could be connected in series and tapped at various points to generate the variety of supply voltages required. This has the disadvantage that the batteries discharge at different rates due to varied current consumption from each tap, and would require a complex charger.
2. A single **lead acid battery** could be used to generate the required supply voltages with the use of small invertors and regulators. This solution is cheaper and more straight forward than using Nicad batteries, but heavier.
3. An **invertor** could be used to step up to 230V which could be stepped down to suitable voltages in the scanner. This approach would have the advantage that the system could be run from the mains power supply when used in a laboratory situation but would likely be too bulky for a portable system and involves considerable power loss.

4.6.4.1 Power Consumption

Power consumption calculations in this section are based on the Mk1 scanner power requirements. The scanner components have the following maximum power requirements:

- Line scan camera : +15V @ 250 mA, -15V @ 250 mA, +5V @ 500 mA
- Mirror stepper motor : + 5V @ 500 mA
- Lens driving : +10V @ 10 mA, -10V @ 10 mA
- IC 's (9 IC's) : + 5V @ ~100 mA

These power requirements are absolute maximum requirements. Actual current consumption has been measured to be considerably lower. For a typical day of system operation the following power requirements have been calculated. These are based on measured current consumption and the assumptions that:

- The system is operated for 10 hours per day
- 100 trees per day are imaged
- Time per image is 100 seconds
- Voltage conversion efficiency of 65%
- Voltage regulation efficiency of 85%

These calculations indicate that for one day of operation powered from two 6V batteries, a battery capacity of 8.8 AHr and 2.0 AHr is required. Batteries of 10 AHr and 6 AHr were used in the power supply.

Voltage	Power required (mWHr)	Conversion efficiency (%)	Battery 0-6V (mAHr @ 6V)	Battery 6-12V (mAHr @ 6V)
+5 V	23250	85	4560	
+15 V	15420	65	990	990
-15 V	4170	65	270	270
+10 V	330	55	25	25
-10 V	330	55	25	25
Safety factor Total			5870	1310
			x 1.5	x 1.5
			8805	1965

Table 4.3 - Scanner power requirements

4.6.4 User Feedback

General description

If a microcontroller based instrument such as the TreeScan scanner is not operating correctly it can be very difficult to get instrument status information. During the design of the system methods should be implemented which will allow the microcontroller code to be debugged and provide status information.

Two features have been implemented to provide feedback to the user on scanner status; Three status LED's, and a serial interface for debugging.

Three **status LEDs** indicate the scanner power status and microcontroller status;

- A green LED indicates the scanner's 5V power is operational.
- A red LED indicates the **power on** state has been entered and high consumption devices are switched on.
- A yellow LED indicates microcontroller status. If the LED is flashing the microcontroller is ready to receive a SCSI command. If the yellow LED is switched on, this indicates the microprocessor is busy and will not accept any commands from the computer.

A **serial RS232 interface** has been implemented to provide a secondary interface between the microcontroller and a computer for the debugging the SCSI interface during development. The microcontroller has an onboard serial interface which generates the required signals. The TTL level serial signal is converted to RS232 serial by the MAX232 line driver.

The serial interface was tested and is operational but it never became necessary to use it to debug the development of the SCSI interface.

4.6.5 Scanner Chassis

General description

The chassis of the TreeScan scanner consists of a 60 cm section of 150 x 100 aluminium channel. This provides a robust chassis inside which other components are mounted. By using this heavy duty channel, component alignment error due to chassis flex is eliminated.

The rotation mechanism is permanently mounted within the scanner chassis. The line scan camera is bolted on to the chassis to prevent possible movement in alignment. The controller board is housed in an insulating plastic casing and bolted to the inside wall of the channel.

The channel chassis is protected by a light sheet metal cover that contains a perspex window, allowing a scan angle of at least 90 degrees (70 degrees used). The scanner is mounted on a tripod during use to provide a steady base for the scanner.

In order to align the scanner with the tree to be measured, a rifle scope has been mounted on the outside of the scanner. This rifle scope is free to rotate in the vertical plane. The scanner is aligned by aligning the scope on the centre of the calibration rod to establish a reference, then tilting the scanner using the tripod head until the scanner is closely aligned with the tree.

All the mechanical development work was completed by Mr Thomas Look.

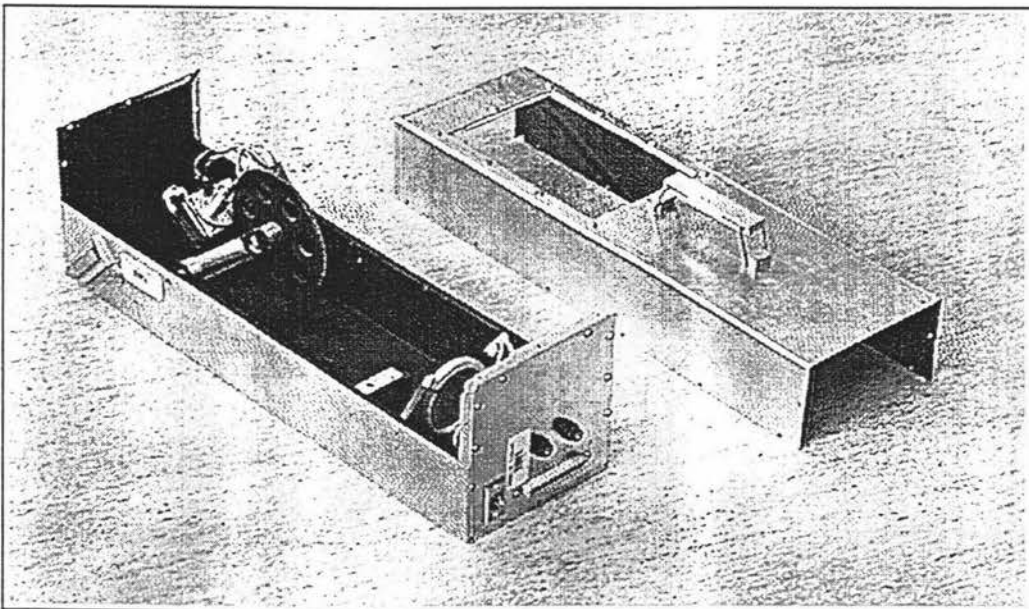


Figure 4.20 - Scanner chassis

4.6.6 Carrying Cases

General description

Carrying cases for the TreeScan system serve two purposes; to enable the system to be carried around, and secondly to protect the system. Two carrying case possibilities have been considered:

- A flexible backpack mounted system
- Sturdy aluminium carrying cases

A backpack carrying case will need to be constructed once the TreeScan system is in operation to aid the ease with which the system can be carried long distances through the forest. However, during the early stages of the development sturdy aluminium protective cases were more important. They were designed to accommodate all scanner components.

4.7 Hardware Development Environment

The hardware development of the TreeScan system involved the development and integration of mechanical components of the system and electronic components of the system. Most of the scanner functionality is under software control so the use of testing software was integral to the hardware development.

The microcontroller development board used during the development was the Mandino Granville 80C517A microcontroller development board. Associated with this development board is the System 51 microcontroller development environment. This provides an integrated development environment that includes a compiler, assembler, microcontroller emulator, and a monitor program.

The scanner electronic hardware was developed and tested in subsections built on veroboard. These subsections could be individually tested without the influence of other scanner components by connecting each section to the microcontroller development board. Once the subsections of the TreeScan controller board had been tested (see figure 4.21), a printed circuit board was designed to accommodate all subsystems.

The development of circuit components on veroboard was very successful, but some noise problems were experienced. The main noise problem was with the use of veroboard for the connection of the SCSI bus from the SCSI controller to the computer over a 1.8 metre SCSI cable.

Protel Technology's Protel Schematic 3 was used to draw up the schematic during circuit design, and Protel Autotrax was used for the printed circuit board layout design (Protel Technology, 1989). The PCB routing was largely manual to facilitate later testing and debugging of the controller board.

A Philips PM3055 60 MHz oscilloscope and a Philips PM3655 logic analyser were used to test signal levels during both the hardware and software development.

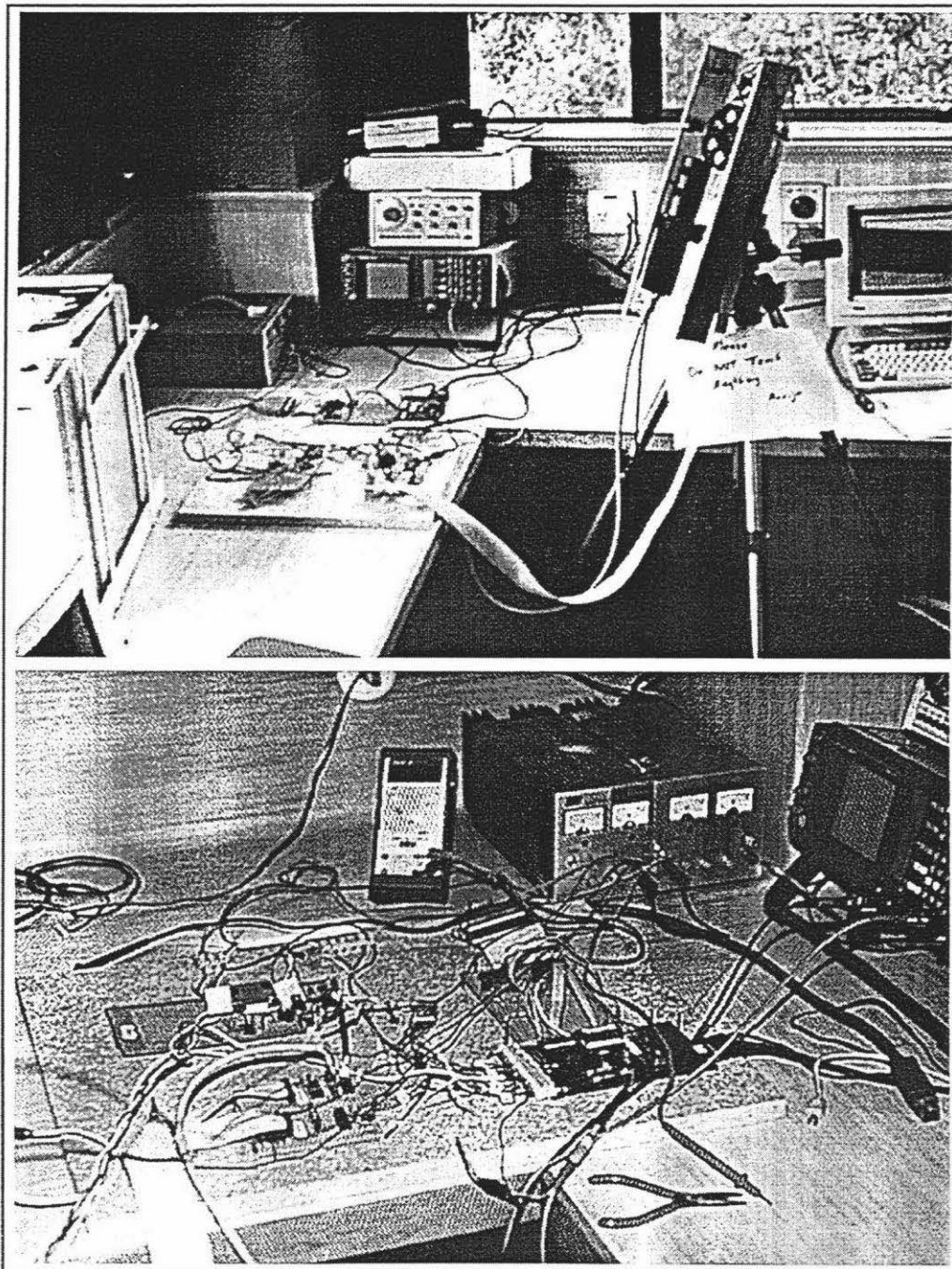


Figure 4.21 - Hardware development environment

Chapter 5

TREESCAN SOFTWARE

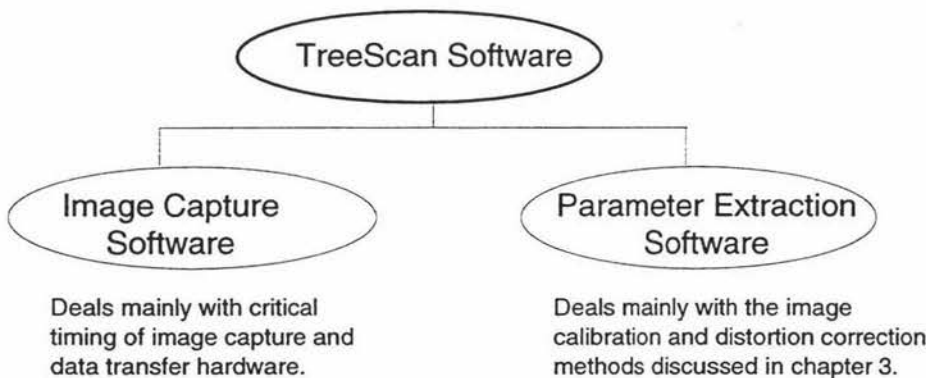
5.1	TreeScan Software Overview	106
5.2	Image Capture Software	108
5.3	Tree Parameter Extraction Software	132
5.4	Software Development Environment	142

The TreeScan software chapter describes the algorithms implemented as part of the TreeScan system. This includes both the **image capture software** which is used to capture images with the scanner, and the **parameter extraction software** which provides facilities to estimate real world tree dimensions from captured images.

The implementation of the algorithms was complicated by the fact that in order to create a system with the functionality of the TreeScan system, software needed to be implemented at four different levels; macros, Pascal, C and assembler. In addition to the functional breakdown there is a physical breakdown with different sections of the same algorithm operating in two different physical locations (see section 5.2.1).

5.1 TreeScan Software Overview

The software developed for the TreeScan system falls into two basic functional categories reflecting the operation of the system; the **image capture software** and the **parameter extraction software**.



The **image capture software** interfaces the computer to the scanner and allows the capture of images using the scanner. This software implements the tasks that need to be done to capture images and deals mainly with critical timing of the image capture and data transfer hardware. The image capture software is distributed across the computer and the scanner, and consists of an acquisition plug-in for the computer and assembly code for the microcontroller.

The **parameter extraction software** provides the facilities to process the captured images and estimate real world tree dimensions. It automates a series of image processing tasks and deals mainly with implementing the image calibration and distortion correction methods described in chapter 3. This software consists of NIH Image macros and NIH Image source additions in Pascal.

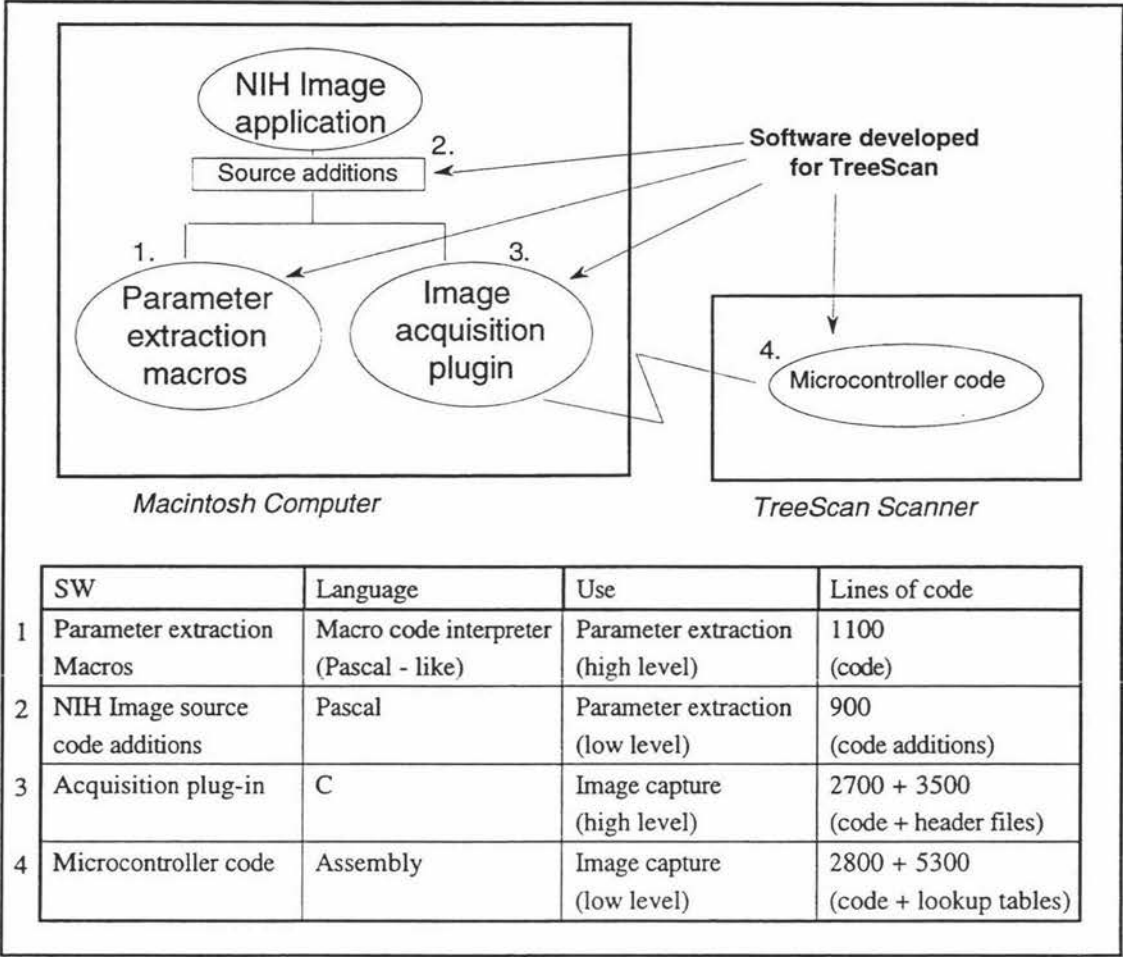


Figure 5.1 - Levels of TreeScan software

The **NIH Image** application has been used as the image processing environment. NIH Image is a public domain image processing package developed by Wayne Rasband at the National Institutes of Health in the USA (Rasband, 1993). NIH Image provides a flexible, easily extendible, and user friendly environment. In addition to this the software is public domain so the full pascal source code is available, and can be modified where necessary. NIH Image has been developed for use with Apple Macintosh computers.

There are **four** levels of software in the TreeScan system, see figure 5.1. A philosophy of implementing all algorithms at the highest possible software level has been adopted to reduce development time. **Image build-up algorithms** have been implemented in the C acquisition plug-in, with low level tasks and time critical tasks implemented in the microcontroller assembly code. **Parameter extraction algorithms** have been implemented in the NIH Image macro language with NIH Image source modifications where additional speed or functionality was required.

5.2 Image Capture Software

5.2.1 Overview

The **image capture software** controls the build-up of images using the TreeScan scanner. This software deals mainly with critical timing of the image capture hardware and critical timing of the image transfer to the computer. The image capture software also provides a straight-forward user interface and the capability to store images for later processing.

The main section of the image capture software is the *image build-up algorithm*. The *image build-up algorithm* implements the sequence of high level tasks required to capture an image; sub-tasks include : performing an autofocus, capturing a section of image, overseeing the SCSI interface, and controlling the scanner hardware (see figure 5.2).

In addition to the functional breakdown, there is a second breakdown that can be made for the image capture software. The image capture software is physically distributed across two locations; the portable computer and the scanner. As a consequence of this, the image capture software consists of two separate programs running on two processors interconnected using a SCSI bus interface.

The image capture software implemented on the **portable computer** consists of an acquisition plug-in. This is a Macintosh code resource which complies with the Adobe

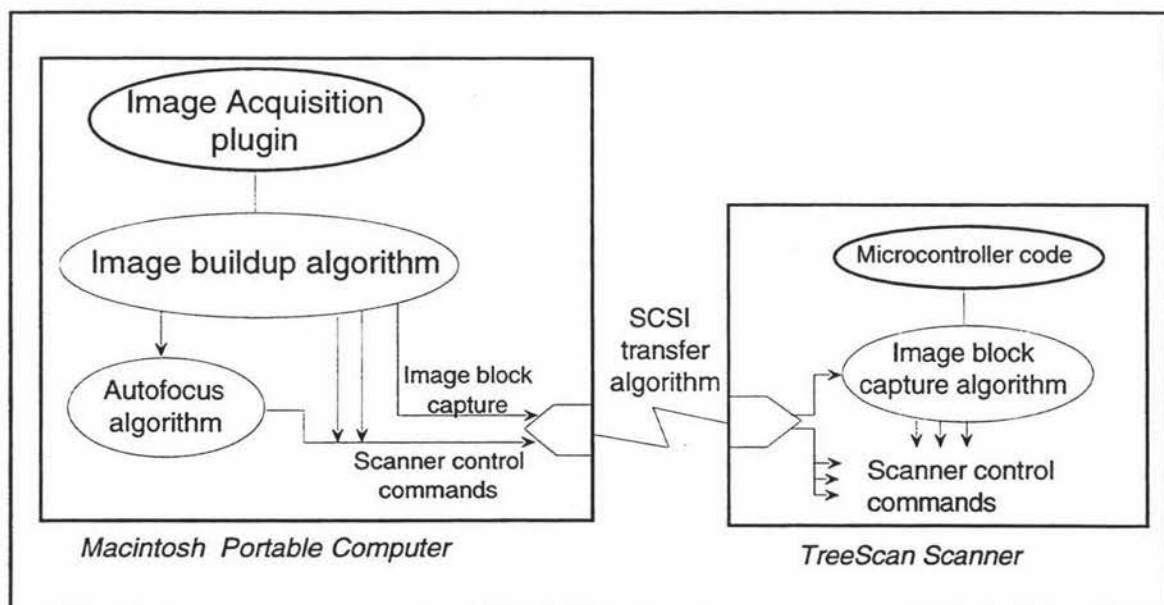


Figure 5.2 - Algorithms implemented in image capture software

interface specifications for plug-in code modules and can be used to extend an application without modifying the base application. Acquisition plug-ins may be executed from any application (typically image processing packages) that supports Adobe format plug-in modules, and are linked to the supporting application at run time.

The acquisition plug-in implements the *image build-up algorithm*, provides a user interface, controls the computer SCSI communications hardware, controls the scanner by sending scanner control commands, and passes a data structure containing the captured image back to the calling application.

The image capture software implemented on the **microcontroller** consists of an assembly language program stored in EPROM. The microcontroller code implements the time critical *image block capture algorithm* and implements the scanner hardware control in response to high level scanner control commands. Assembly language executes fast but it takes considerably longer to implement and debug a complicated algorithm than it does in a higher level language. The microcontroller code provides the simple building blocks that can be put together to perform the required action by using a series of commands.

After a brief discussion of software constraints, key algorithms are outlined in sections 5.2.2 to 5.2.5. Finally, the TreeScan image capture software is further discussed in sections 5.2.6 and 5.2.7.

5.2.1.1 Image capture software design constraints

It was decided to maintain system flexibility and use the microcontroller to perform the A/D conversion of the image data (see section 4.2). This led to timing constraints on the design of the software in the areas of:

- **A/D conversion of image data** (line scan camera timing)
- **SCSI transfer timing**

The A/D conversion is a synchronous process; the SCSI transfer is an asynchronous process. In a normal situation involving synchronous data processing in operations such as this, at least one of the operations would be interrupt driven with buffering of data in RAM, however this approach would reduce overall speed.

There is not sufficient time to implement an interrupt based approach at maximum A/D conversion rates. Instead a dedicated hardware loop has been implemented (see section 5.2.3.1). The A/D conversion must operate at a fixed rate while image data is clocked out of the camera. The SCSI interface can transfer data at a rate 40 times that of the A/D conversion. To speed up the image capture the *image block capture algorithm* was designed to perform the A/D conversion and to send the bytes directly to the SCSI bus

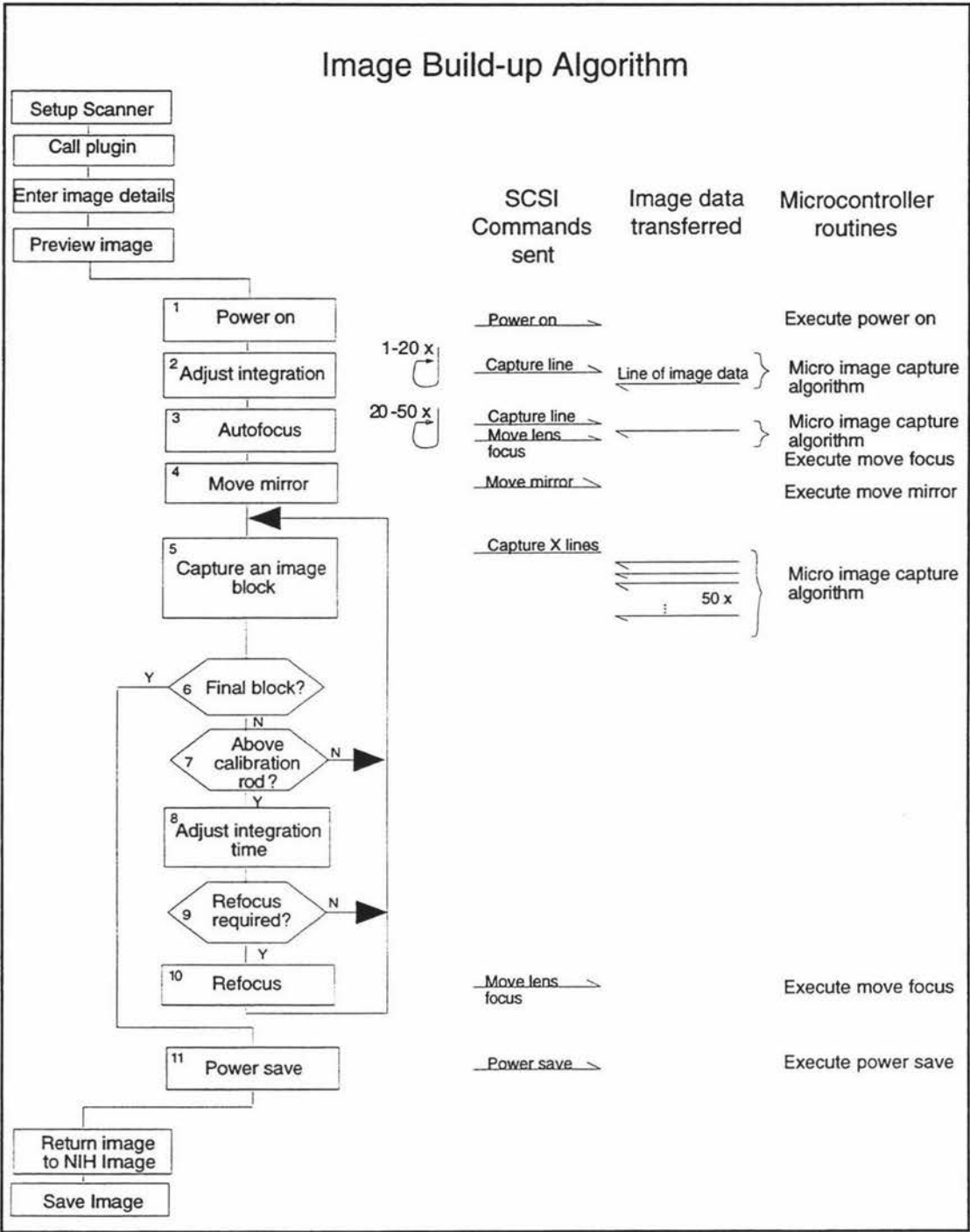


Figure 5.4 - Image build-up algorithm

The integration time for the current block and the number of lines to be captured is sent to the microcontroller as part of the command descriptor block. This allows the microcontroller image block capture algorithm to capture the required number of lines and to modify the integration time without the need for additional SCSI instructions.

It was decided that an integration time adjustment should be made often. Integration time adjustments during the scan are calculated on the previous image data block and do not require additional lines to be captured.

1. Power is turned on to the high current components by sending a **PowerOn** command to the scanner.
2. The integration time is adjusted for the current light level by repeatedly capturing one line and adjusting the integration time until correct exposure is achieved. Up to 20 iterations of the loop are performed depending on lighting.
3. An autofocus is completed to focus the base of the tree at the start of image capture. This consists of finding the focus position by repeatedly capturing a line, calculating a measure of focus, and adjusting the lens focus position until the optimal focus position is found. This would typically involve 20 to 50 iterations of the loop.
4. The mirror is stepped down by 500 steps so the image capture begins at the base of the tree.
5. A loop is entered which captures blocks of 50 lines until the number of lines to be captured has been reached. A single image capture command is sent to the scanner and 50 lines of image data are received back. The timing involved for the capturing of each line is carried out by the microcontroller image block capture algorithm. The computer simply receives bytes until the correct number of image bytes have been received to fill the block.
6. If the correct number of blocks have been received, exit the image capture loop.
7. If still within the calibration rod area of the image, a refocus or integration time adjustment is not required.
8. Adjust integration time for the next block based on the pixel values of the previously received block. No commands need to be sent to the scanner as the integration time is sent to the scanner as parameters in the image block capture command.
9. & 10. If a focus adjustment is required, call the blind refocus routine to estimate correct focus position and send a command to the microcontroller to move the lens focus to the desired position.
10. Perform the above loop until the desired number of lines have been captured.
11. Turn the power off to the high current components by sending a **PowerSave** command to the scanner.

Figure 5.5 - Image build-up algorithm (description)

It has been decided to make 32 focus adjustments per complete image as this allows the tree to remain within the scanner depth of field without an undue increase in image capture time.

Once the image is captured it is passed from the plug-in to NIH Image to be saved. The image can now be further processed by the parameter extraction software.

5.2.3 Image Block Capture Algorithm (Microcontroller)

In the *image build-up algorithm* the capture of an image line or block of lines consisted of a single scanner control command sent to the microcontroller. This task of capturing one or more lines is the *image block capture algorithm*, consisting of a complicated sequence of events with critical timing aspects.

The *image block capture algorithm* performs the A/D conversion of the image data and transfer to the computer using the SCSI interface. In addition, the camera timing signals are produced, the mirror stepper motor is stepped between lines and a SCSI transfer error detection and correction method is implemented. These are time critical elements so they have been implemented in the microcontroller.

When the microcontroller receives the command from the computer to capture a block of lines, the sequence of steps shown in figures 5.6 and 5.7 is executed.

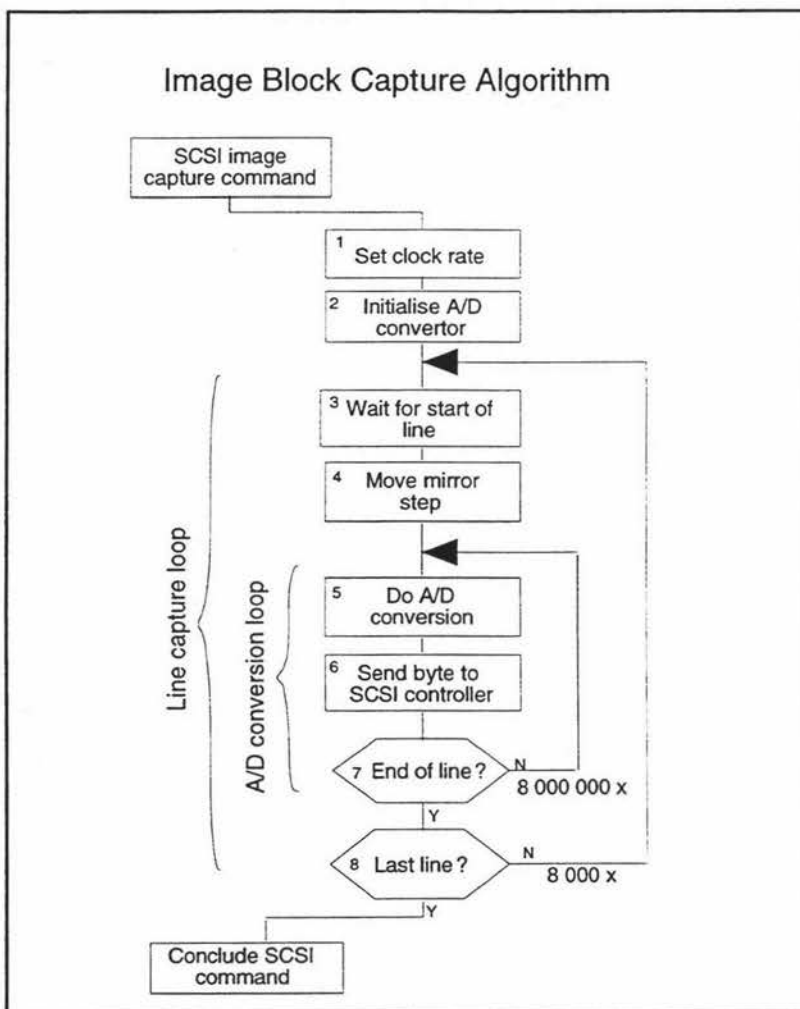


Figure 5.6 - Image block capture algorithm

It is critical that the timing within the A/D loop is optimised as this loop is executed 8 million times for an 8000 line image. The line timing must also be optimised but this depends on a number of factors and is less critical.

The time required to capture a block of lines depends on two main factors, either of which can be the limiting factor: the data conversion and transfer rate, and the integration time (exposure time). Data conversion and transfer and image integration take place in parallel. Data conversion and transfer takes a fixed period of time. Integration time depends on the lighting conditions.

In normal daylight conditions the integration time is an order of magnitude shorter than the data conversion and transfer time, so the image capture time is limited by the data conversion and transfer time. If however the scene illumination is low, the situation may be reversed and the image capture time may be limited by the integration time.

5.2.3.1 Dedicated Conversion and Transfer Loop

The minimum time to perform an A/D conversion is 7 μ s for the microcontroller used. The guaranteed interrupt service time of the microcontroller is 7 μ s. If interrupt based routines were used this would slow down the system.

In a normal situation with synchronous data processing operations such as the A/D conversion of image data, at least one of the operations would be set up to interrupt the processor with the buffering of data in RAM.

Instead a dedicated hardware loop was implemented as there was not sufficient time to implement an interrupt based approach.

1. The number of lines to be captured is initialised from the command descriptor block, and the frequency of the camera clock signals is set based on the integration time information in the command descriptor block.
2. The A/D converter is initialised to run continuously triggered on the DATA RATE signal returned from the camera.
3. Wait for the LINE SYNCH signal indicating the start of a line. Image data will be valid after 24 DATA CLOCK cycles.
4. Move the mirror position so the next line can be exposed.
5. & 6. Enter the A/D loop. Digitise the analog voltage signal and send each byte to the computer over the SCSI interface.
7. Loop for 1024 pixels in each image line.
8. Loop for the number of lines in this block.

Figure 5.7 - Image block capture algorithm (description)

5.2.3.2 Image Capture Line Timing

The data conversion and transfer time takes a fixed 12 ms per line (assuming 8 bit A/D conversion and that a write to RAM is used). In parallel with this is the 2 mS delay to allow the stepper motor to settle and the variable length integration time as shown in figure 5.8. The camera sensor integration starts on the falling edge of the LINE/INTEGRATE signal, and the rising edge indicates the start of a new line.

If the integration time is shorter than the 10 mS (daylight conditions) the data conversion and transfer time becomes critical and it takes 0.6 seconds to capture a 50 line block. The start of integration is delayed so that the end of integration is synchronised with the end of the line instructions.

If however the integration time is longer than 10 mS (low light conditions) the situation is reversed and the integration time becomes critical. The integration starts at the end of the stepper settling time and the start of the next line is delayed to accommodate the integration time. The time to capture a 50 line block will be variable and dependent on lighting. The integration time maximum has been set to 120 ms (10000 DATA CLOCK cycles) which means it takes 6 seconds to capture a 50 line block (15 minutes / image)!

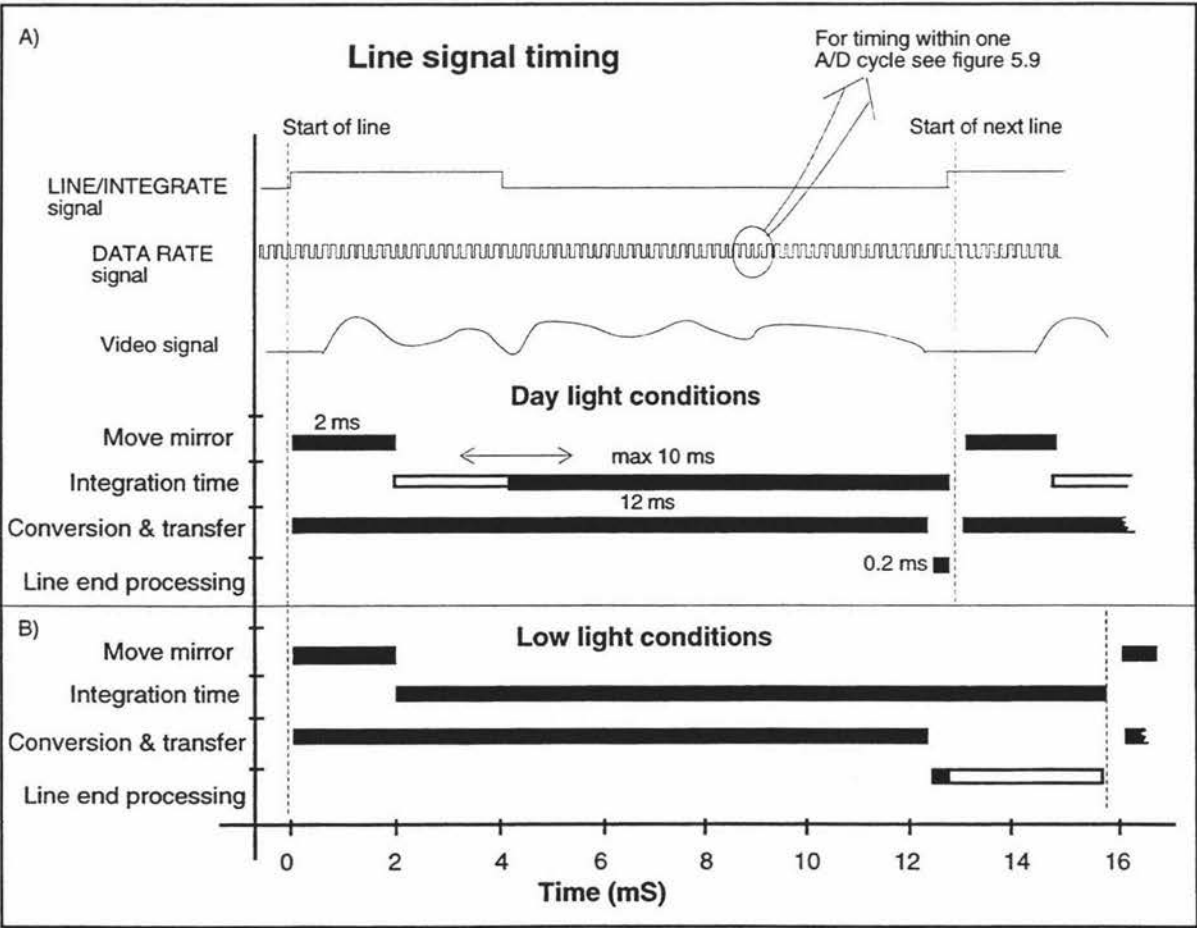


Figure 5.8 - Line signal timing

5.2.3.3 Image Capture A/D Conversion Timing

In the section on image capture line timing it was stated the data conversion and transfer time are fixed. This is not strictly true as these parameters depend on the implementation of the A/D loop. The data conversion and transfer time consists of 1062 cycles of A/D loop time. This section describes the timing within a single DATA CLOCK cycle.

The A/D converter is triggered by the falling edge of the DATA CLOCK after which the result of the A/D conversion is ready after 7 μ s. The result of the 10 bit conversion is stored in two registers. The top 8 bits are stored in the ADDATH register and the bottom 2 bits are stored in the ADDATL register. The data must be read from these registers before the next A/D conversion can start. After the pixel data has been read, the microcontroller must process it before the next A/D conversion is completed.

The minimum time that the A/D loop can take is indicated in figure 5.9a. After the result of the A/D conversion (top 8 bits of the converted value) has been read, it is transferred to the SCSI controller, a check is made to determine if the end of line has been reached,

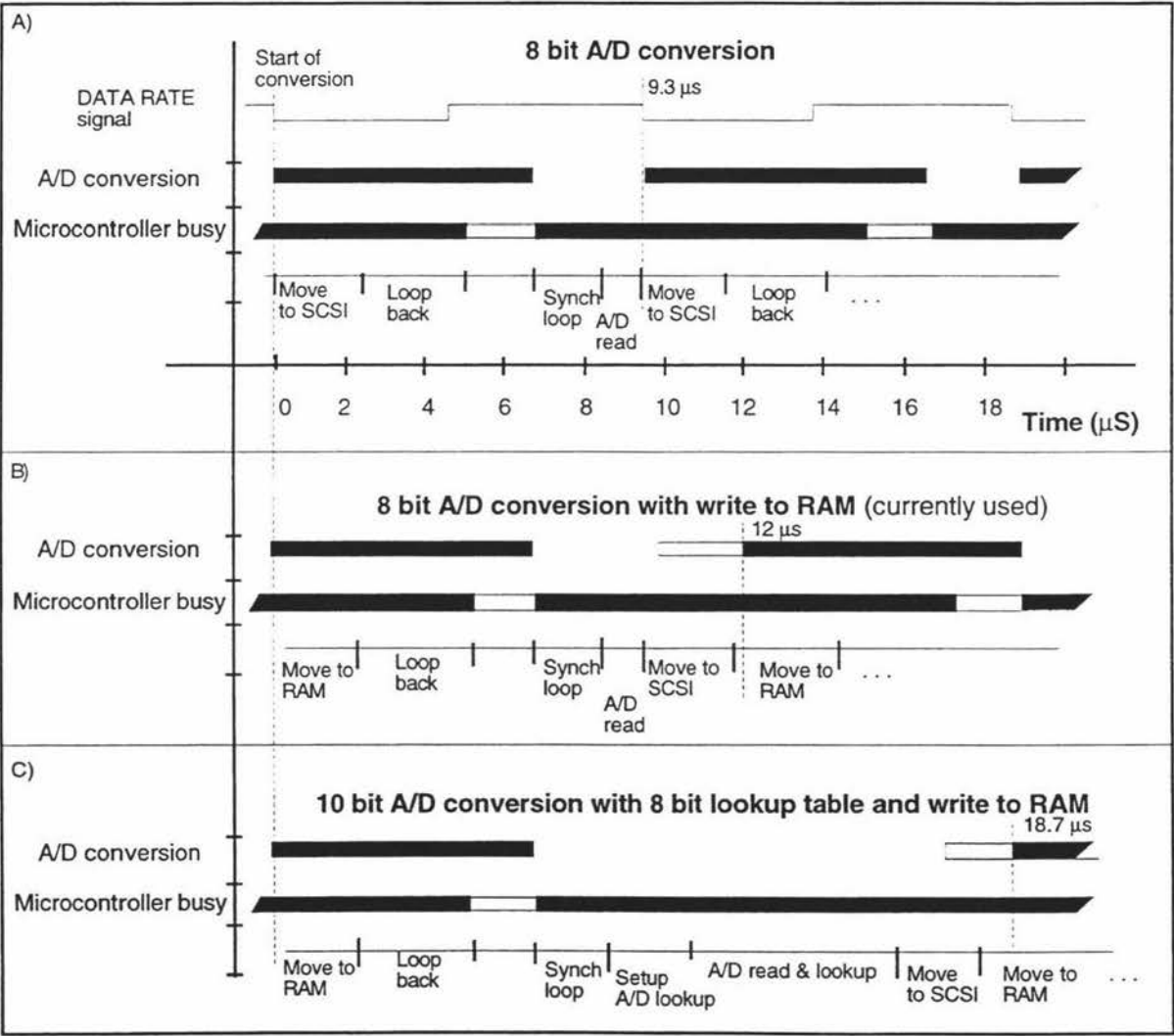


Figure 5.9 - A/D signal timing

```

; *****
;   Get into loop capturing data and sending it to SCSI
; *****
r_L8_ad:  nop                                ;1
          nop                                ;1
r_L8_ad2: jb    BSY,r_L8_ad2                ;Wait until AD complete ;2
          mov   A,ADDATH                     ;read data out         ;1
          mov   DPSEL,#sxDPTRFIFO           ;1
          movx  @DPTR,A                     ;copy data to SCSI      ;2

          mov   DPSEL,#sxDPTRBuffer         ;1
          movx  @DPTR,A                     ;buffer data to RAM     ;2
          inc   DPTR                         ;1

          djnz  crCountL,r_L8_ad             ;2
          djnz  crCountH,r_L8_ad2           ;1
          ;---
          ;16 +4

```

Figure 5.10 - A/D conversion (8 bit) microcontroller code

if not the microcontroller loops back and waits for the next value. See figure 5.10 for the main A/D loop and its timing in machine instruction cycles (0.75 μ s).

Two other A/D conversion routines have been implemented to cater for low lighting conditions and SCSI related transfer problems (see sections 6.4.1 and 5.2.4.3). In both situations the minimum time for each A/D conversion is lengthened, increasing the minimum time required to capture a block of lines.

Several small timing reductions could be made to the A/D conversion process, however in most forest conditions the data conversion and transfer time is not the critical time so it was deemed more important to deal with low light levels. One improvement that could be made is to have the end of line count interrupt driven instead of counter driven. This would reduce A/D loop time by 1.5 μ s (or approximately 10% speed increase).

5.2.4 SCSI Transfer Algorithm

The SCSI transfer algorithm implements the interface between scanner and computer. It allows the computer to send control information to the scanner and allows the scanner to return the captured image data.

Previously it has been stated that the microcontroller executes tasks based on scanner control commands passed to the scanner from the computer. Each scanner control command consists of a SCSI command sent by the acquisition plug-in to the microcontroller using the SCSI bus controllers in the Macintosh and the scanner.

- At a **logical level** a SCSI command consists of the transfer of a command descriptor block. A command descriptor block is a data structure containing a command operation code and parameters associated with the opcode. Depending on the command opcode the SCSI command may involve further data transfer in either direction.
- At a **software level** a SCSI command involves of a series of chip commands being sent to the SCSI bus controllers which perform the correct sequence of bus phase changes to facilitate the transfer of the command descriptor block and the image data.
 - In the **computer** this requires a sequence of Macintosh Toolbox calls to the SCSI Manager. These calls operate directly on the computer SCSI bus controller.
 - In the **scanner** this is completed by reading and writing to registers in the scanner SCSI bus controller. This involves both sending the correct chip commands and transferring any data in and out of the SCSI data buffer.
- At the **electrical level** a SCSI command consists of a complex sequence of control signals on the nine SCSI bus control lines. The signal lines indicate bus phase status, and perform byte handshaking as data is transferred in either direction over the nine data lines.

The detailed requirements of the SCSI protocol can be found in appendix I and a description of the scanner SCSI bus controller hardware in section 4.4.

In this section implementation of the SCSI transfer algorithm is discussed. A normal SCSI transfer will be described after which the problems encountered during the SCSI software implementation and their solution are discussed.

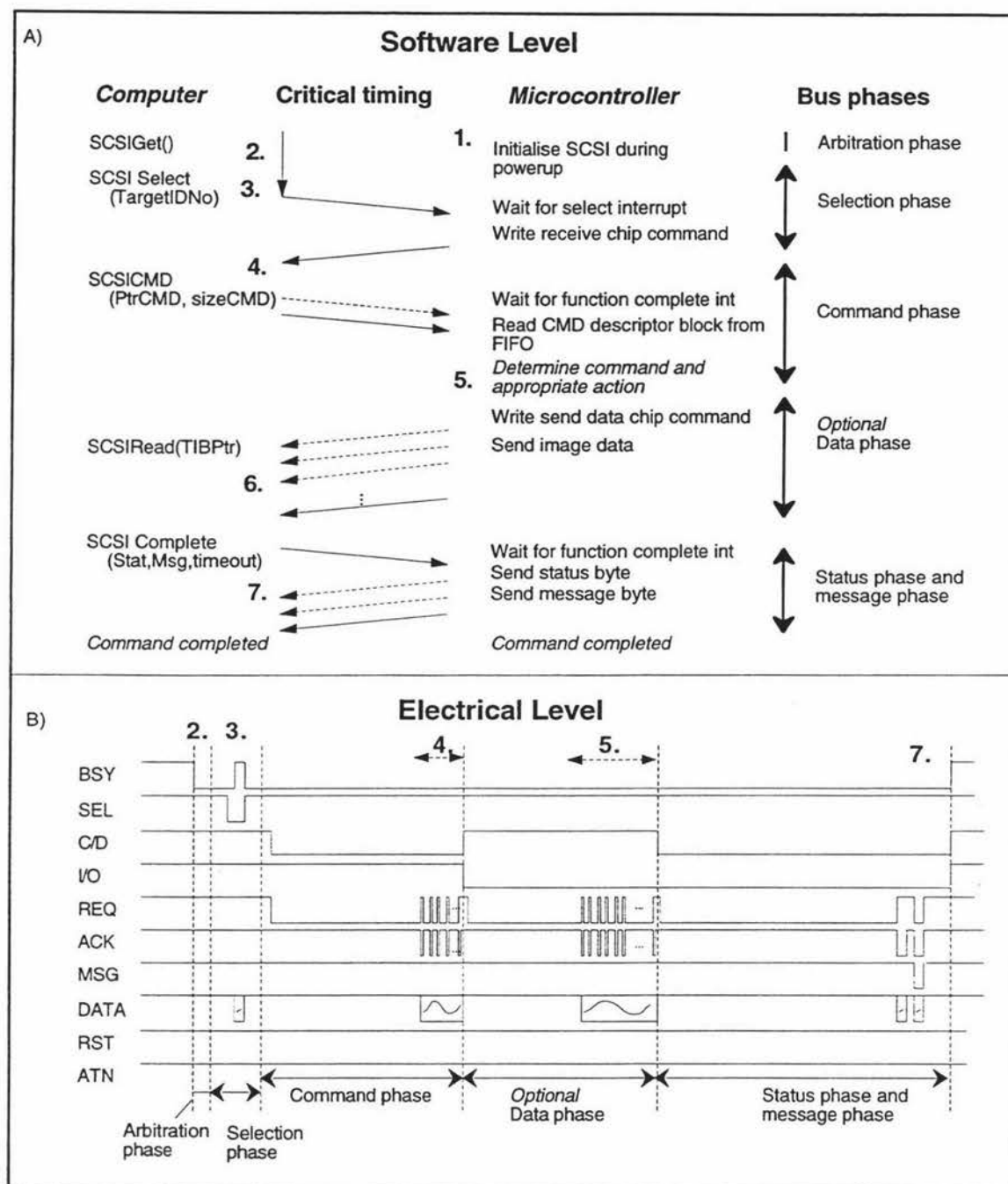


Figure 5.11 - Normal SCSI transfer

5.2.4.1 Normal SCSI Transfer

A normal SCSI transfer involves the sequence of operations shown in figures 5.11 and 5.12. The operations completed by both the plug-in and the microcontroller software are listed with the critical path highlighted. The SCSI bus phases throughout the transfer are also shown.

The computer must select a target device (the scanner) and wait for it to respond. Once the device responds the computers sends the command descriptor block to the scanner.

1. When the scanner is powered up, the scanner SCSI bus controller is initialised. This includes the setting of SCSI ID number and enabling the selection of this device as a target. Now the scanner waits until it is selected.
2. When a SCSI command transfer is started, the computer must first arbitrate for the SCSI bus. This is performed through a SCSIGet() toolbox call and allows the computer to become an initiator and gain control of the SCSI bus.
3. Next the target device must be selected using a SCSISelect() call, with the target device SCSI ID number as a parameter. When the scanner's SCSI ID number is selected the microcontroller is interrupted with a select interrupt and the RECEIVE COMMAND chip command must be written to the scanner SBC. This acknowledges to the computer the scanner has been selected and is ready and waiting to receive a command descriptor block.
4. The command descriptor block is transferred from the computer to the scanner using the SCSCMD() call with a pointer to the command descriptor block as a parameter. The information is transferred and arrives at the scanner with a function complete interrupt. The command descriptor block data is read out of the SCSI bus controller FIFO. The appropriate action is performed based on the information in the command descriptor block.
5. If required the data phase is entered. The data phase involves the microcontroller sending data and the computer receiving it. Data is sent from the microcontroller by writing a SEND DATA chip command to the SBC, writing the number of bytes to be transferred to a counter, and writing the bytes to be transferred to the SBC FIFO. At the computer the data transfer is performed using a SCSIRead() call with a TIB (Transfer Instruction Block) as a parameter. A TIB is a sequence of low level instructions for the computer SCSI bus controller (see appendix K). The TIB must contain the number of bytes to be read.
6. If the number of bytes in the TIB, in the scanner SBC counter, and the actual bytes transferred do not match, synchronisation is lost and both the scanner and the computer can lock up waiting for each other to transmit data or change the bus phase. A timeout can be set that limits the maximum time the computer will wait while locked up.
7. When the data transfer is completed the computer makes a single SCSIComplete() call. The microcontroller sends a status byte and a message byte. This involves both the status phase and the message phase. Once the status byte and message byte have been transferred the SCSI transfer is completed. The SCSI bus returns to the bus free phase and the scanner waits for the next SCSI command.

Figure 5.12 - Normal SCSI transfer (description)

The scanner completes tasks based on the command descriptor block with the optional transfer of data (see figure 5.12, item 5). The command must then be completed by the transfer of two bytes to indicate whether the transfer was completed successfully.

At the electrical level this involves the bus signalling indicated in figure 5.11b. The BSY signal is activated at the start of the transfer, is briefly switched low during the selection phase and remains active throughout the duration of the transfer. The state of the C/D (Command/Data) and I/O (Input/Output) lines determine the bus phase. During the

region marked 4 the command data is transferred and during the region marked 5 the image data is transferred. The transfer of each byte of data has an REQ/ACK handshake associated with it.

If the SCSI command does not involve image transfer the data phase is simply omitted from the sequence.

5.2.4.2 Obstacles Encountered implementing SCSI

Two significant problems encountered during the development of the SCSI transfer algorithm were caused by unexpected timing fluctuations in the operation of the SCSI interface:

- 1 . A byte loss problem as a result of transfer buffer overflow
- 2 . Long delay times between the changing of SCSI bus phases

A byte loss detection and correction algorithm was implemented to deal with both the byte loss problems. Images were captured in blocks of 50 lines per SCSI command to reduce the impact of long delays during bus phase changes.

Byte loss

As discussed in previous sections (4.4.3, 5.2.1.1 and 5.2.3.1), although SCSI is a handshake system with a 'wait if not ready' flag, there is insufficient time to check this flag so a dedicated loop converts image data and sends it to the SBC. This assumes the SCSI interface (4 MBytes/s) is able to keep up with A/D conversion (100 kHz). Small variations in transfer rates would be handled by the built in 32 byte buffer and hardware handshake.

Sometimes however, the computer SCSI bus controller was unable to receive data for intervals up to 1.5 mS in duration. This delay, probably due to operating system background tasks, allowed image bytes to be lost as image data can only be buffered for

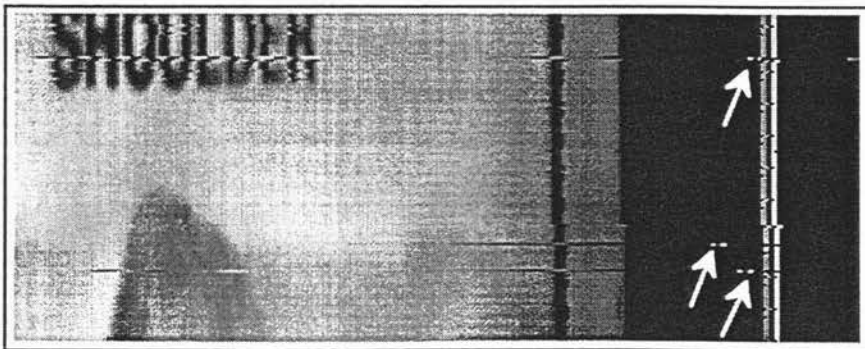


Figure 5.13 - Image with byte loss problem

a maximum 0.5 mS by the 32 byte FIFO. As the image conversion continued, this allowed the FIFO to overflow and image bytes were lost. Figure 5.13 shows a section of image captured with bytes lost in three places, as can be seen on the white alignment reference on the right of the image (see figure 5.13).

There are several possible solutions to this problem:

1. Ideally the SCSI bus controller FIFO should be checked before any bytes are sent. If however the FIFO is full the A/D conversions need to be delayed. This is not possible without losing camera timing synchronisation.
2. The capture and transfer of image data as two separate interrupt driven processes (as discussed in section 5.2.1.1). This increases the conversion and transfer time resulting in considerably longer image captures.
3. A byte loss and detection scheme can be set up, as delays are rare, to resend the line from memory (checking whether FIFO is not full) if the some of the image bytes were lost during transfer. This requires the image bytes to be stored to memory during capture, but this results in minimal A/D time increase.

The final method has been implemented and is discussed in greater detail in the next section.

An interesting aspect of the byte loss problem was that the delay in receiving bytes was more common on the portable Macintosh PowerBook 520c data acquisition computer than on the Macintosh Quadra, on which the plug-in was initially developed. On both computers all extensions were switched off to remove as many background tasks as possible without hacking into the operating system. This did not make a significant difference in the frequency of the problem.

Minimum SCSI command time

The second obstacle encountered was that the Macintosh SCSI bus controller appeared to be slow to react to SCSI phase changes (see figure 5.14). There was a delay of approximately 26 ms from the time the microcontroller wrote the RECEIVE COMMAND chip command to the time the command descriptor block data was received back from the computer. The same delay was present before the computer acknowledged the first byte of the data transfer, and a longer delay of 55 ms was present between the end of the data phase and the transfer of the status byte.

These delays were expected to be three orders of magnitude smaller, similar to the arbitration and selection delays of approximately 20 μ s. The cause of these delays should be further investigated and is still not fully explained.

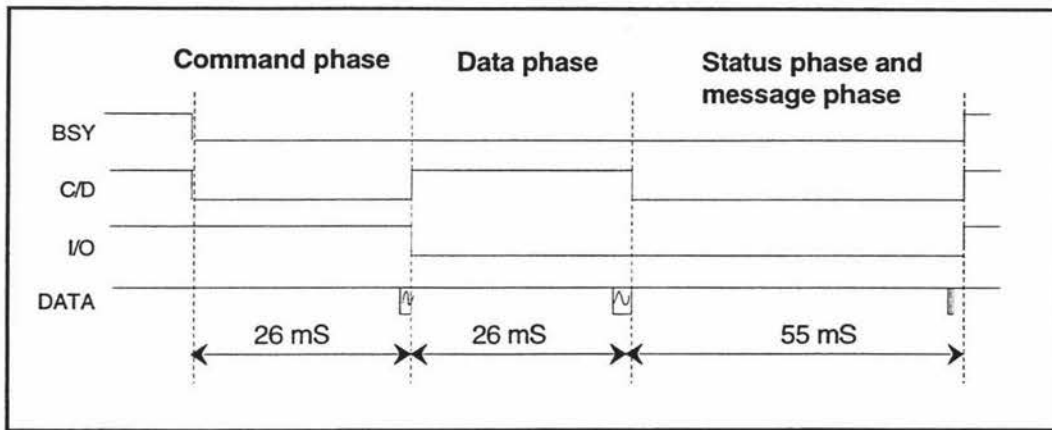


Figure 5.14 - Extended delays during SCSI transfer

These delays in the changing of bus phases introduced a minimum duration for a SCSI transfer of approximately 120 ms. If lines were captured one line per SCSI transfer this would allow the maximum transfer rate to be 8 lines per second. By capturing the image in blocks of 50 lines this minimum transfer delay becomes less significant and lines can be transferred at an average rate of 70 lines per second which is close to the maximum data rate of 83 lines per second.

5.2.4.3 Byte Loss Detection and Resend Scheme

The byte loss detection and resend scheme is an elaborate scheme to ensure transferred data is not corrupted by buffer overflow. It will correct for occasional buffer overflow of up to several hundred bytes. An overview of the byte loss detection and resend scheme is presented here, with details presented in appendix L.

There are two restrictions that must be taken into account:

1. The TIB instruction set is very limited and can only execute seven types of instructions (see APPENDIX K for more detail on TIBs).
2. The scanner SCSI bus controller provides limited status information. There are flags that indicate whether the Transmit FIFO is full or half full, but not whether the transmit FIFO is empty.

The detection and resend scheme works on the basic principle that the SBC expects to send a certain number of bytes. If at the end of the image line the SBC expects to send more bytes, the SBC FIFO must have overflowed during the A/D conversion loop. The line that was sent must be ignored and the line resent from memory.

A block of image lines is captured using a single Macintosh SCSI call. This means that the detection and resend scheme must be implemented within a single TIB. The TIB uses self modifying code to conditionally execute instructions. The microcontroller

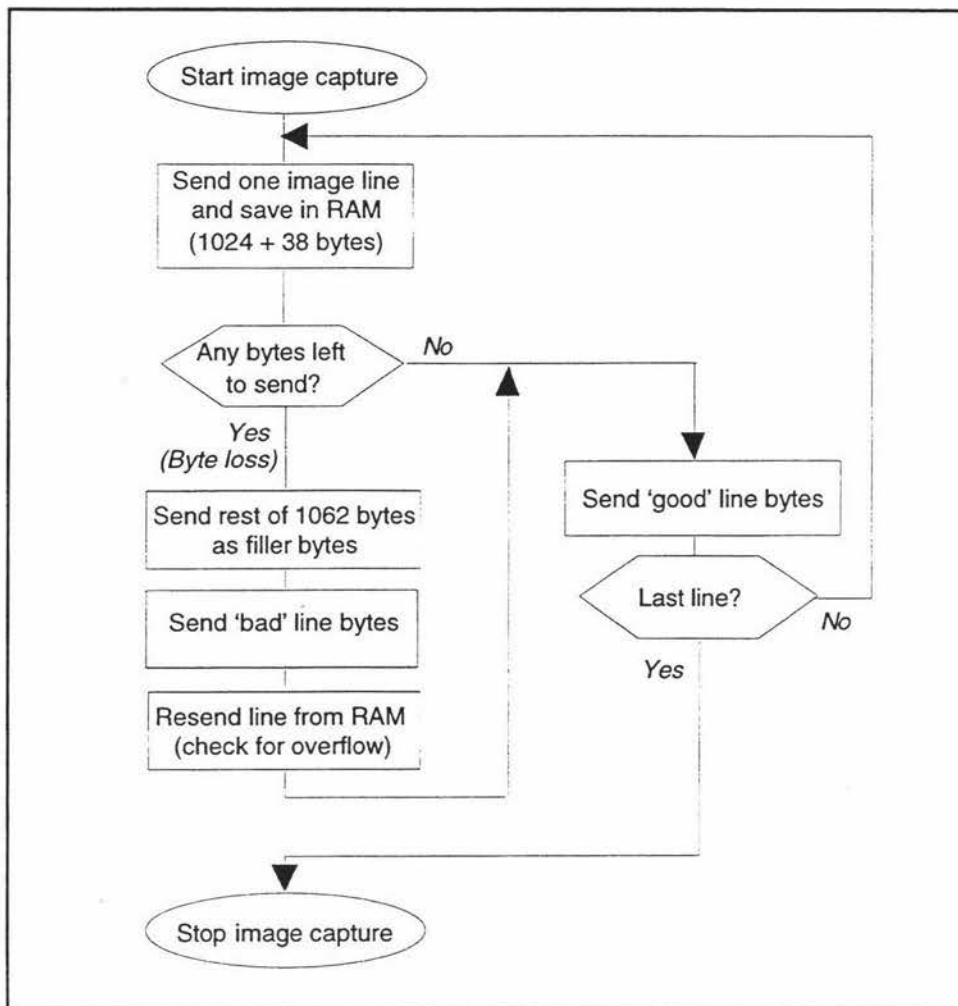


Figure 5.15 - Byte loss detection and resend scheme

instructs the computer to overwrite the last line by modifying the TIB and transferring an increment to the pointer in memory where the image is stored.

The implementation of this byte loss detection and resend scheme has little impact on the acquisition plug-in code. Only the TIB must be redeveloped to allow for significant self modification.

The implementation of this byte loss detection and resend scheme does have an effect on the image capture timing. The A/D conversion loop must write the image data to RAM as well as to the SCSI. This increases the A/D conversion loop to 12 μ s as shown in figure 5.9b. The capture image block algorithm (figure 5.6) has also been modified to accommodate the additional write to RAM, additional end of line checking, and a line resend if necessary.

5.2.5 Focus Algorithms

There are two aspects to focusing the scanner; the scanner must initially focus on the base of the tree before the image capture starts, and correct focus must be retained throughout a scan.

5.2.5.1 Focus Adjustment

During an image capture scan the distance to the object changes, so the focus position must be adjusted. Three focus approaches can be adopted:

- The **focus can be set halfway up the tree** so the whole of the tree remains within the depth of field and no focus adjustment is required. This requires a very small aperture and thus a large exposure time.
- An **autofocus may be made every time a change in focus is required**. A smaller depth of field, and hence shorter integration time, could be used as a series of focus adjustments may be made up the tree. This method requires that the autofocus algorithm has been implemented and could focus in the wrong place if branches obscure the tree.
- A **method of 'blind focus' may be implemented**. This method completes an autofocus at the bottom of the tree which is used to approximately determine the distance away from the tree. Based on this distance away from the tree, the estimated focus position is calculated for subsequent focus points. Throughout the scan the lens is moved to the estimated point of focus. Branches will not affect this method of refocus up the tree. This method relies on absolute positioning of the focus and the assumption that the tree is approximately vertical.

It was decided to adopt the third approach to retain focus throughout the image capture. This involved the development of an autofocus algorithm, a 'blind refocus' algorithm and a repeatable lens control mechanism.

5.2.5.2 Autofocus Algorithm

The basis of an autofocus algorithm is that it must find the point of optimal focus, which is taken to be the lens position at which the measure of focus defined in chapter 3 is at its maximum.

Two autofocus algorithms were implemented which operated on a slightly different principle. The Mk1 lens system provided servo control only for the focus position of the lens, with the result of this was that absolute position information could not be used to

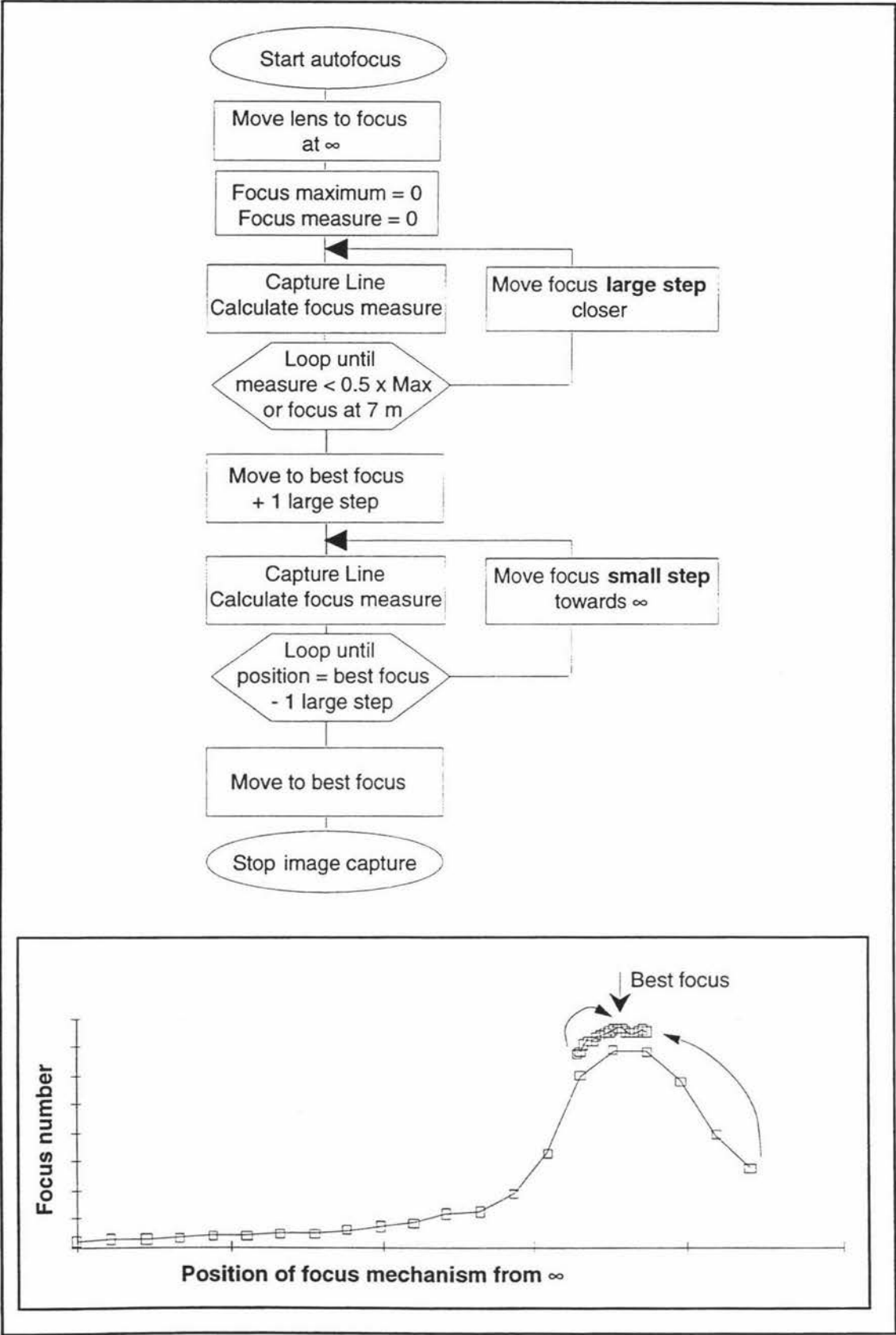


Figure 5.16 - Final autofocus algorithm

position the lens focus. Stepper motor control of the lens focus position was provided on the Mk2 system. This allowed the focus algorithm to make use of absolute positioning information, and allowed the blind refocus algorithm to be implemented.

The final focus algorithm implemented on the Mk2 system is shown in figure 5.16. The microcontroller drives the lens to its infinity position. Next the lens is driven towards near focus in large steps. At each point a line is captured and focus number calculated. Once this number has reached a maximum and is getting smaller the optimal focus point has been passed. Next the algorithm searches around the best focus point it found, in smaller steps to find the optimal focus point to a high precision. Finally absolute positioning is used to move to the optimal focus point.

The Mk1 autofocus algorithm was based on a similar principle in that it drove forwards until over the focus peak, then reversed direction and drove back in smaller steps until the focus number started going down again. This is based on the assumption the curve is a monotonic rising and falling curve. If there is any noise or variation in calculated focus numbers this algorithm may not find the optimal focus point.

In developing the autofocus algorithm it is very important to ensure the focus calculation is based on a region of the image where all the imaged objects are the same distance from the scanner. If this is not the case the focus graph will be a bimodal curve and it will become difficult to find the point of best focus. All focus algorithms are based on a single line in the image because it would be time consuming and introduce unnecessary wear on the mirror tilting mechanism to capture several lines and rapidly drive back and forth between them.

An evaluation of the autofocus algorithm is presented in section 6.4.4.

5.2.6 TreeScan Plug-in Software

The scanner control software implemented on the portable computer consists of an acquisition plug-in. This is a Macintosh code resource which complies with the Adobe interface specifications for version 3 plug-in code modules and may be used to extend applications.

The TreeScan acquisition plug-in implements the image build-up algorithm, and provides a user interface so the operator can control image capture. The plug-in controls the scanner by sending scanner control commands. Each command is sent by interacting directly with the computer SCSI bus controller. The TreeScan image capture plug-in passes a data structure containing the captured image back to the calling application (NIH Image in this case).

The TreeScan plug-in is programmed in C. It captures the image into memory during the *start* selector call. This allows the image to be displayed in a preview window as it is being captured. To be able to capture an 8000 line image, the plug-in requires 8 megabytes of memory. After the image is captured in memory it is passed to NIH Image in small blocks during the *continue* selector call.

The portable computer has 20 megabytes of RAM. To capture an 8000 line image

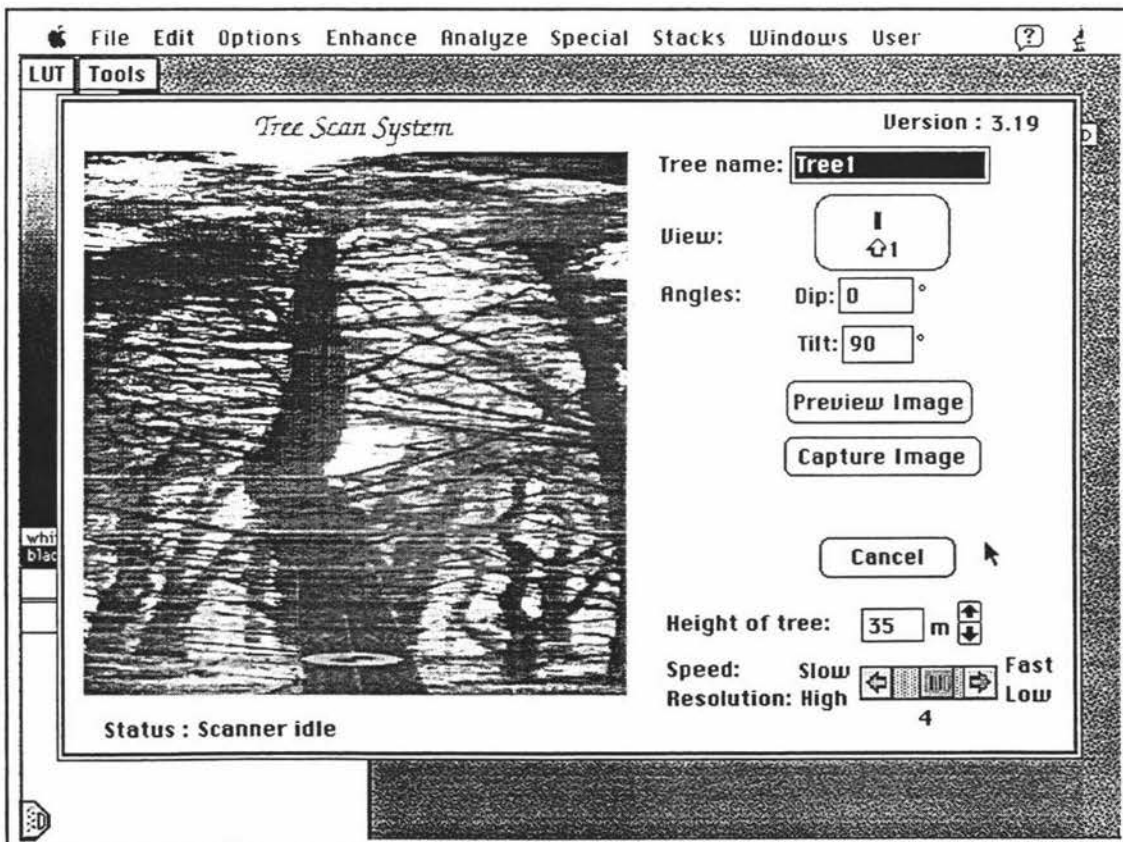


Figure 5.17 - TreeScan image capture user interface

requires two 8 megabyte image buffers; one for the plug-in and one for NIH Image to store the returned image. After these two buffers are allocated this leaves 4 megabytes. This is just enough for the operating system, NIH Image application, and memory required for the plug-in code.

NIH Image also allocates a clipboard and undo buffer. In order to call a plug-in each needs to be the size of the image. This would require a further 16 megabytes. To avoid this the NIH Image source code was modified to deallocate the clipboard and undo buffers directly before an acquire plug-in call and reallocate the buffers directly after the acquire plug-in call. This frees 8 megabytes of memory which NIH Image can use during the image capture.

The TreeScan plug-in source code consists of six source files (2700 lines of code) with associated header files and library files. A total of 6200 lines of code. Relevant source listing are provided in appendix M.

The user interface consists of a dialog box which is presented when the plug-in is called (see figure 5.17). The dialog box has a number of button controls and an image window in which a small scale version of the captured image is displayed during capture. The buttons allow the user to set up the tree information, preview a small section of the image and capture a full image. A debug dialog is also implemented which can be used to execute individual scanner control commands to aid development and testing of the hardware.

The image build-up algorithm (see section 5.2.2) is executed when the **Capture Image** button in the dialog box is pressed (see figure 5.17).

5.2.6.1 Adobe Plug-in modules

As explained a plug-in module is a compiled Macintosh code resource which may be used to extend applications. Plug-ins are designed to complete specific image processing tasks and must comply with the Adobe interface specifications. Plug-ins are linked to a supporting application at run time and may be executed from any application (typically image processing applications) that supports Adobe format plug-in code extensions.

Adobe Photoshop version 2.0 supports three types of plug-in modules; acquisition plug-ins, export plug-ins, and filter plug-ins. NIH Image also supports all of these.

The requirement for an acquisition plug-in module is that it conforms to the Adobe interface specification as described by the documentation on writing plug-in modules (Knoll, 1991). Certain resource types must be correctly set and the plug-in should be called using the following Pascal calling conventions:

```
PROCEDURE PlugIn ( selector: INTEGER; acqRec: Ptr;  
                  VAR data: LONGINT; VAR result: INTEGER);
```

The plug-in must respond to the following sequence of **selector** values:

1. *Prepare* : allows the plug-in to adjust its memory allocation
2. *Start* : returns the parameters of the image being captured to the calling application and allows the plug-in to display it's dialog box
3. *Continue* : returns a section of an image to the calling application
4. *Finish* : allows the plug-in to free any required memory

The **acqRec** parameter is a pointer to an *AcquireRecord* structure which contains image information such as: the maximum memory available, image x and y dimensions, and a pointer to the image data area in memory. The **data** parameter is used by the plug-in as a pointer to its global data, and the **result** parameter allows the plug-in to return its status.

5.2.7 Microcontroller Software

The microcontroller code controls the scanner hardware and implements the time critical tasks. Hardware control is provided by responding to a series of scanner control commands (SCCs) passed from the acquisition plug-in. The image block capture algorithm is time critical and implemented in the microcontroller code. The microcontroller code was written entirely in assembly language.

The structure of the code reflects its function. The microcontroller code consists of a background loop waiting for SCSI commands from the computer. When a SCSI command is received, the appropriate routine is called to perform the required function. The microcontroller then returns to the background loop waiting for the next SCSI command.

The microcontroller source code consists of seven source files (2800 lines of code) with four extra files containing the 10 bit A/D lookup tables. A well structured set of naming conventions was set up to keep the use of variable names and constant names consistent within the assembly code.

The main file TASM120.ASM contains important code documentation. This includes microcontroller I/O port declarations, memory map, register usage, variable naming convention, modification history, constant declarations, and variable declarations. Relevant sections of the source listings can be found in appendix M.

5.3 Tree Parameter Extraction Software

5.3.1 Overview

The **parameter extraction software** provides the facilities to process captured images and to estimate real world tree dimensions. This software automates a series of image processing tasks and deals mainly with implementing the image calibration and distortion correction methods described in chapter 3.

The parameter extraction software can be used in a variety of ways dependent on whether the images are just being browsed to determine the size of various features of interest or a systematic analysis of each image is being undertaken. Immediate feedback is provided for interactive processing. In addition key information such as the three dimensional stem model can be stored in a form suitable for later processing and stem breakdown optimisation.

The processing of captured images is divided into three broad tasks:

- 1 . Image calibration
- 2 . 2D processing
- 3 . 3D processing

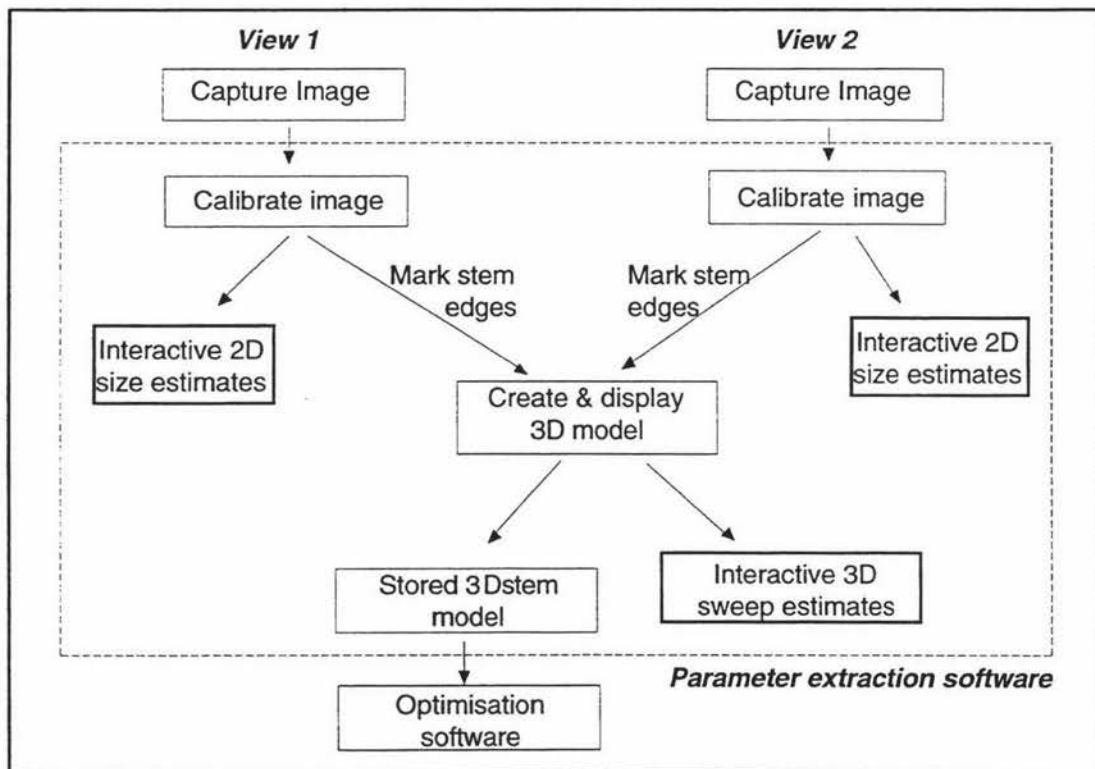


Figure 5.18 - Parameter extraction sequence

The image must first be calibrated to establish a relationship between the image dimensions and real world dimensions. Once the image has been calibrated, size estimates in two dimensions can be interactively made from a single image, or information from two calibrated images can be combined to estimate three dimensional shape.

The two dimensional tree size estimates allow any dimensions on the 2D calibration plane (see section 3.2) to be estimated, these include: heights, stem diameters, branch diameters, internodal distance and other distances between features.

The **three dimensional processing** generates a three dimensional model of the tree stem shape. Interactive sweep estimates can be made on this stem shape model, and this model can be further processed by optimisation software to determine the optimal stem breakdown.

The parameter extraction software consists of a number of macros written in a Pascal like macro programming language which is a feature of NIH Image. The macro language is used for the majority of the processing tasks. In some cases however where the macro language is too slow or lacks functionality, additional user routines have been added to the NIH Image application. This involves modifying the NIH Image source code and recompiling the application.

In sections 5.3.6 and 5.3.7 the structure of the TreeScan parameter extraction software is described in greater detail.

5.3.2 Image Calibration

As discussed in section 3.5.1, image calibration is required to establish the geometry of the image capture situation before real world dimensions can be estimated from the image. Once the image capture geometry is established, the distortion correction methods can be used to estimate the size of features up the full length of the tree.

During image calibration points on the calibration rod in the image are marked establishing the image capture geometry.

Two different methods of image calibration have been implemented dependent on the distortion correction method used:

- The first distortion correction method used was based on estimating the camera position based on the calibration rod with two crossbars. This involved marking six points on the image. From these six points the distance and angle θ was determined. This method was found to be imprecise (see section 3.5.2.2) and not further developed.

- The second distortion correction method used was based on estimating distance from the scanner to the tree using a calibration rod with only one crossbar and measuring dip and tree lean angles. These angles are measured using a digital level and entered into the plug-in dialog box which saves the data to file. Three points on the bottom crossbar are marked.

Once the image is calibrated the information is available to estimate the real world position of any point on the image. The calibration data is also saved to file. This allows the image to be re-calibrated later without having to mark the points on the calibration rod again.

The distortion correction method implemented is the 'TreeScan perspective correction - measured O' as described in section 3.5.3.2.

It should be noted that the software does not modify the entire image to correct for perspective distortion as this would involve considerable processing. Instead, only the coordinates of each point of interest are converted to real world coordinates. These may then be used to estimate tree parameters.

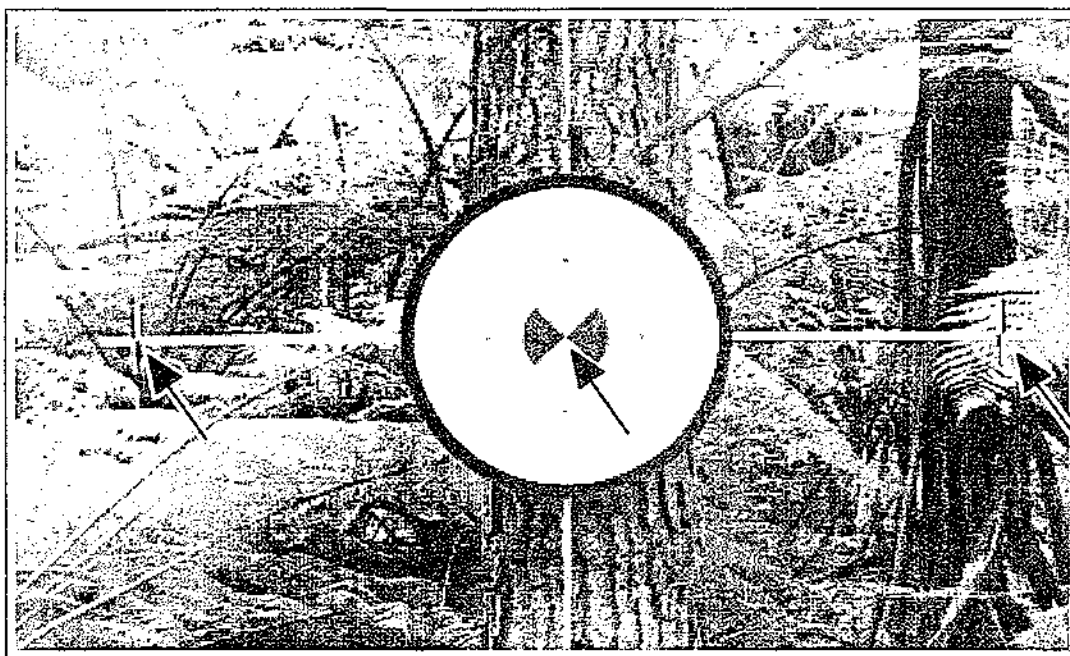


Figure 5.19 - Marking of three calibration points

5.3.3 Feature Size Estimation in Two Dimensions

The two dimensional feature size estimation involves the estimation of feature sizes from a single calibrated image. This allows any dimensions on the 2D calibration plane to be estimated, these include: heights, stem diameters, branch diameters, internodal distance and other distances between features.

Two dimensional feature size estimation involves three steps; identifying the feature of interest on the image, marking the feature of interest, and estimating the size of the feature of interest.

1. Features of interest must first be visually identified on the image. This may be more difficult than visually identifying features in the forest because two fixed views of each tree are captured. Some features or branches may be hidden in both views. Zoom and pan are available to change the displayed resolution. Features may be marked at any resolution.
2. Once identified, the feature must be marked by making a line selection for distance estimates, or placement of the cursor for height estimates. The selection must be manually made using judgement to correctly place the end points of the line or crosshair pointer (see figure 5.20).

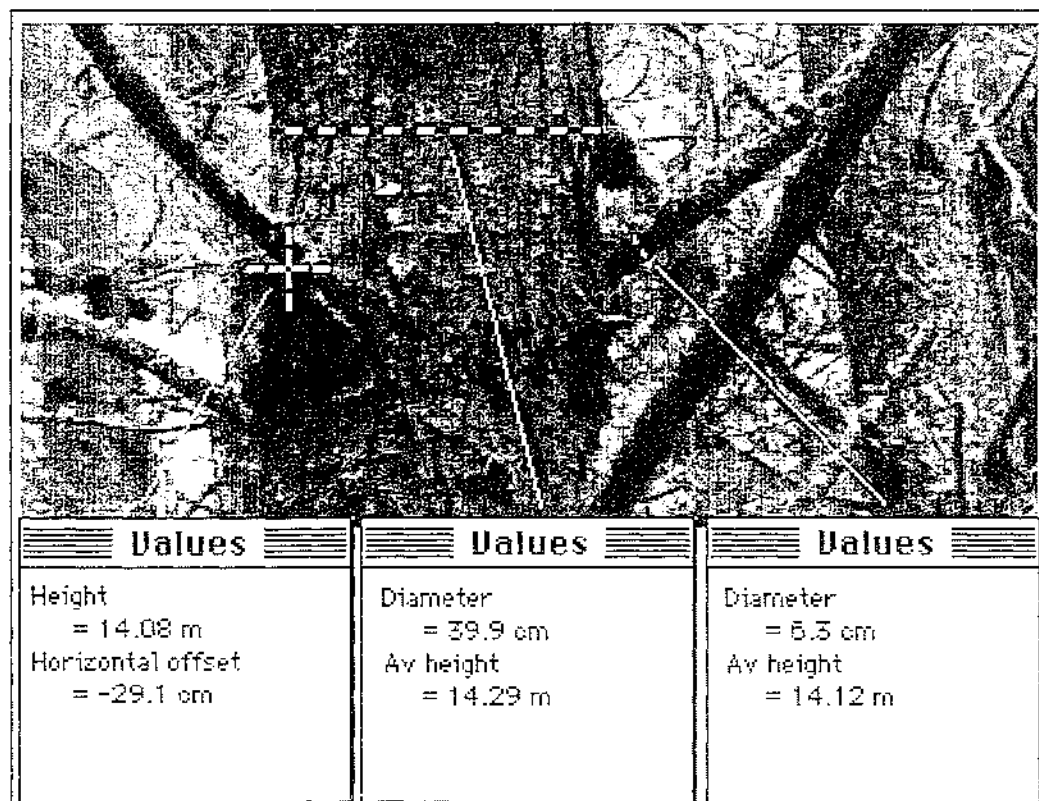


Figure 5.20 - Two dimensional feature size estimates

3. When the feature of interest is marked, the software processes the coordinates of the selection, correcting for image capture distortion and calculating a size estimate of the feature. Cursor placement provides height and offset from the calibration reference, while line selections provide a distance between end points of the selection.

This sequence allows the dimensions of any tree feature to be estimated provided that it is visible in the image. The main disadvantage of this method of identifying features is that estimates depend on the manual placement of the cursor. In the estimation of branch diameters for example, the branch may only cover a few pixels. Research is currently underway by Dr Ralph Pugmire and Mr Ian Overington to automate some or all of the parameter extraction processing and to allow estimates to subpixel accuracy.

5.3.4 Three Dimensional Stem Shape Estimation

To estimate sweep, the position of the tree stem in three dimensional space must be measured. This is completed by generating a model that is defined by the three dimensional tree shape.

The generation of this three dimensional model consists of a process different from two dimensional feature estimation. Information from two views of the same tree must be combined, and rather than individual feature heights and offsets, shape information is required for the full length of the stem. Position estimates are made for sufficient points up the edge of the stem to fully define a three dimensional stem model.

The processing required to generate the three dimensional model consists of a sequence of tasks shown in figure 5.21. The marking of the stem edges must be performed manually. This task is tedious but must be performed carefully as the dimensions of the tree stem model are based on the placement of these lines. The automatic detection of these tree edges is another task currently being investigated as part of the automatic parameter extraction research being undertaken by Dr Ralph Pugmire.

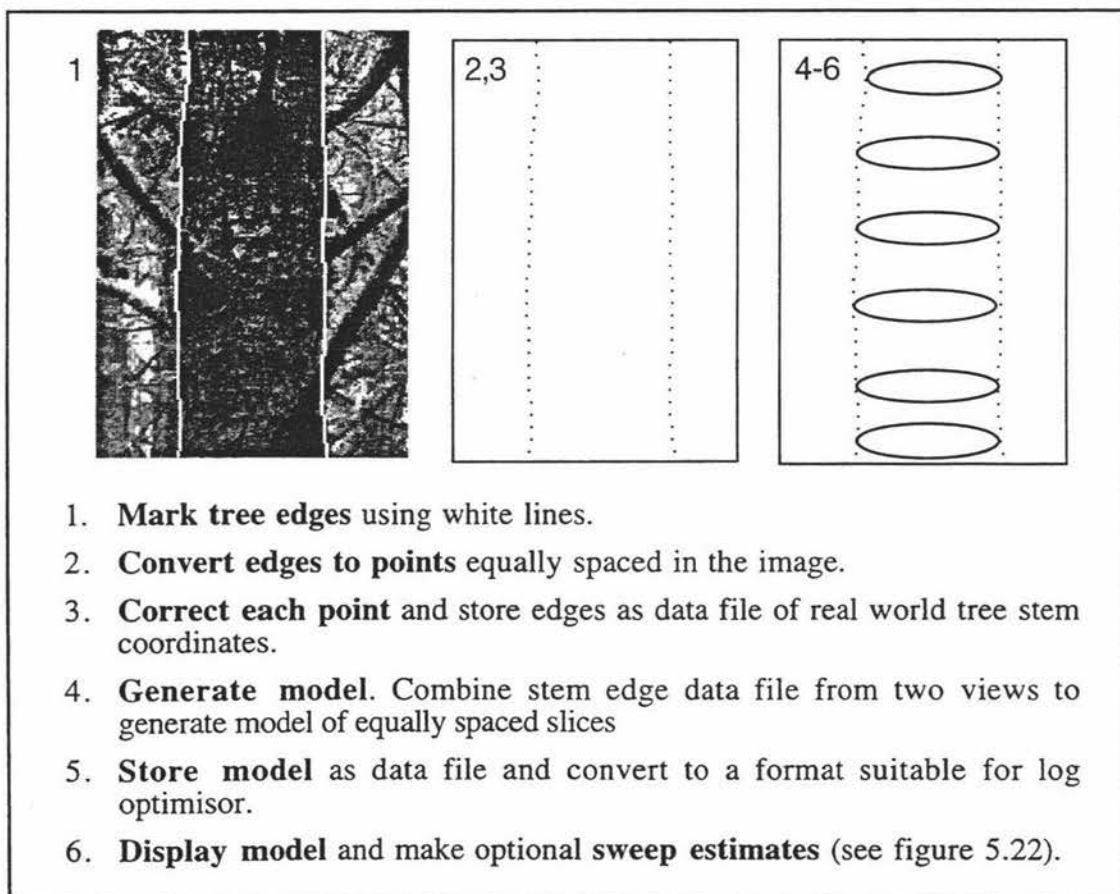


Figure 5.21 - Generation of three dimensional stem model

Once the edges of the stem have been marked these are processed into a series of points, which are corrected to produce a file of stem shape coordinates. At this stage the model consists of points that are at equal pixel spacing in the image, or increasing spacing up the tree.

The stem shape files from both views are processed to generate the three dimensional stem model. The stem data is interpolated to generate a model which consists of 'slices' spaced at fixed distances up the tree stem. This stem model is saved and converted to a format readable by the log optimisation software developed by Tasman Forestry in conjunction with Auckland University (Tasman, 1995). This software has been developed for the optimisation of felled tree stems on the skid site.

The stem model can be displayed and rotated. Using the displayed model representation interactive sweep estimates can be made for variable or fixed length sections of the tree stem.

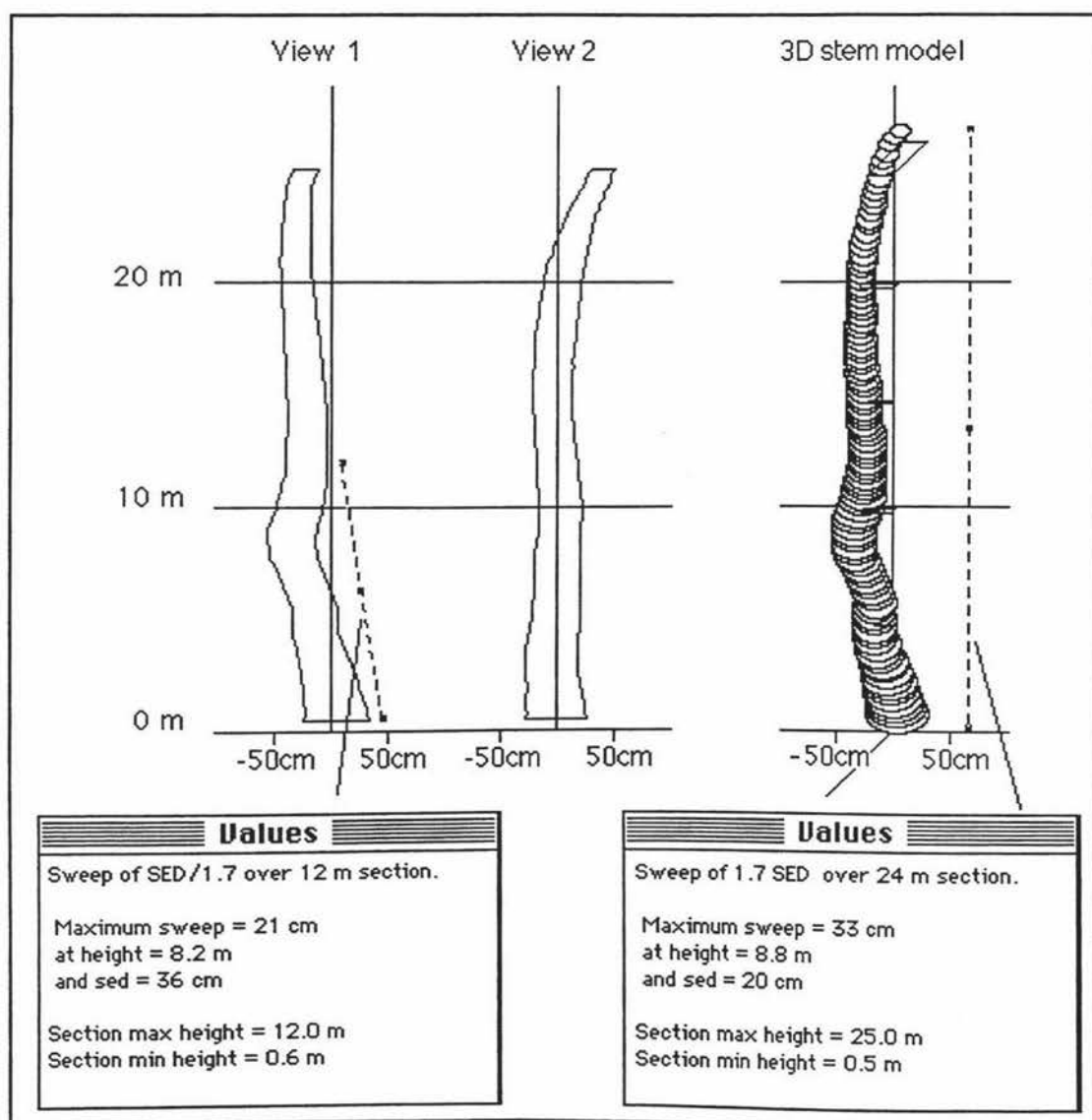


Figure 5.22 - Sweep estimation from displayed tree model

5.3.5 Possible Improvements to Parameter Extraction

The methods currently implemented are capable of generating all the required tree size estimates. However this processing is a tedious semi manual task during which the operator must mark all dimensions to be estimated. Research is currently underway by Dr Ralph Pugmire and Mr Ian Overington to automate some or all of the parameter extraction processing and allow estimates to be made to subpixel accuracy.

There are two other ways in which the accuracy of parameter estimates could be improved:

- The optimisation software determines the optimal stem breakdown based primarily on stem shape and branch sizes. The stem model currently passed to the log optimiser only contains stem shape information. Branch information should be added to the 3D tree stem description. This would provide a more useful system to determine optimal stem breakdown. This is currently being implemented.
- In the two dimensional size estimation only information from one view is used to calibrate the image. If the tree is in front of or behind the calibration plane an expected error is introduced as described in section 3.7. The software could be modified to combine the stem position information from the second view to reduce this introduced error. If implemented the estimation of tree dimensions becomes an iterative task with a small improvement in estimates.

5.3.6 TreeScan Macros

The TreeScan macros are a series of routines that perform the tasks discussed in the sections on image calibration (section 5.3.2), two dimensional feature size estimation (section 5.3.3) and three dimensional tree shape estimation (section 5.3.4).

The TreeScan macros are written in a Pascal-like macro programming language that is a feature of NIH Image and can be used for automating complex or repetitive tasks. Loaded macros are accessible through the normal Macintosh menu interface or may be assigned to special key strokes. NIH Image macros consist of a text file that can be edited using the NIH Image in-built text editor. The macros can be easily modified and can be loaded by NIH Image at any time the application is active. The macros are interpreted at run time eliminating the need for software to be recompiled.

The maximum size of the macro file is 32 kByte in size. The implemented macros are larger than 32 kByte, so TreeScan macros have been split up into two files. The main file contains the normal processing macros. The second file contains a large number of utility macros.

Main TreeScan macros	TreeScan Utility Macros
F1 - Display help screen	P - Print 50% image in A4 pages
F2 - Capture image	CMD P - Print full image in A4 pages
F3 - Load image	F - Filter to remove banding
F4 - Remove white pixels (initialisation)	D - Vertically decimate image
F5 - Calibrate image	T - Extract tree region
F6 - Save edges to file	B - Extract base region
F7 - Perform 3D conversion and save model to file	M - Extract target region
F8 - Display 3D model	Q - Setup image thumbnail
F9 - Display height	A - Paste left side thumbnail
F10 - Display diameter	S - Paste right side thumbnail
F11 - Display sweep	...
F12 - Draw scale on image	
CMD M - Load utility macros	CMD M - Load TreeScan macros

Figure 5.23 - TreeScan processing and utility macros

5.3.7 NIH Image Source Additions and Modifications

In situations where the macro language was too slow or required additional functionality, additions have been made to the NIH Image source code to extend the application.

NIH Image is written in Pascal and consists of 34 source files. It has been designed to allow additional routines to be added in the *user.p* source file. This file has the calling structures in place so that a call to extra procedures can be made using the `USERCODE()` call with the correct parameters. The complete application needs to be recompiled if changes are made.

The main routines added were routines for the processing of data files and the processing and display of the 3D model. This includes the saving and loading of the calibration file, combining of two stem data files into the 3D tree model, and displaying the tree model.

Several modifications have also been made to the NIH Image application. The main modification is the de-allocation of the undo and clipboard buffers before an acquisition plug-in is called and the reallocation afterwards. This allows large images to be captured by freeing up as much memory as possible for the image capture (see section 5.2.6).

5.4 Software Development Environment

Each software level required development in a different programming language and a different development environment.

ThinkC version 6.0 was used to develop the TreeScan plug-in, ThinkPascal version 4.0 was used to modify the NIH Image source code, and ThinkReference version 2.0 was used as the main source of reference information for the Macintosh managers and the toolbox calls. Inside Macintosh volume I - VI were used as important references, as well as reference information specifically for plug-in development and SCSI development (Knoll, 1991).

The majority of the SCSI development was performed on a Macintosh Quadra with 8 megabytes of RAM. The Quadra has a separate SCSI bus controller for external devices which provided greater safety during development as SCSI errors on the TreeScan system could not affect the internal hard disk. Once operational development was performed using a Macintosh Powerbook 520c. This portable computer is the computer that is used for the in forest image capture and contains 20 megabytes of RAM and a 160 MByte hard disk.

All microcontroller software development was completed using the Mandino Granville monitor software version 4.43 and assembler version 3.08 which purchased with the 80C517A microcontroller development board.

Chapter 6

TREESCAN EVALUATION

6.1	Overview of Evaluation	144
6.2	Sequence of Evaluation Experiments	145
6.3	Hardware Calibration	147
6.4	TreeScan Characterisation	151
6.5	Initial Accuracy Tests in Two Dimensions	157
6.6	Final Accuracy Tests in Two Dimensions	158
6.7	Accuracy Tests in Three Dimensions	160

This chapter is an evaluation of the TreeScan system. It reports on the system capabilities and discusses the modifications necessary to convert the scanner as originally designed and built to an accurate scientific instrument.

The evaluation of a prototype such as the TreeScan system is a cyclical testing process. Each time around the cycle more knowledge is gained as problems are solved and modifications are made to the system. Once modifications are made many of the previous tests need to be repeated to ensure the results are still valid. The material in this chapter has been collated into a logical sequence of experiments to establish the system capabilities.

6.1 Overview of Evaluation

An evaluation of a system such as the TreeScan system should go through several stages. The system hardware must first be calibrated and characterised. Only once this has been completed can the actual accuracy of the instrument be evaluated.

The initial scanner prototype is the Mk1 system. A series of tests were completed to check the calibration and characterisation. Next experiments were completed to check the accuracy of TreeScan object size estimates. This successfully identified several serious weaknesses of the Mk1 system:

- Calibration procedure was inherently imprecise
- Insufficient light under forest conditions
- Lens modification required

A second prototype (Mk2 version) was developed to overcome these weaknesses. The calibration procedure for the second prototype was modified, and a different lens with larger aperture was purchased to provide more light and solve several lens problems.

A second series of experiments was performed. These experiments showed the accuracy of the Mk2 system to be good and within the required specifications. Experiments completed include hardware calibration, characterisation experiments, two dimensional accuracy tests, and three dimensional accuracy tests.

Once accurate and repeatable results were attainable in both two and three dimensions less critical functions such as the final user interface received attention.

6.2 Sequence of Evaluation Experiments

An overview of the evaluation experiments performed is presented here in chronological order. This overview is provided to set the context for the results of important evaluation experiments presented in subsequent sections. The conclusions of each experiment are presented with the sequence of modifications and further experiments necessary to quantitatively establish the accuracy of the TreeScan system.

Once the TreeScan system was developed to the stage that images could be reliably captured, an evaluation of the system was started. At this stage an autofocus algorithm had been implemented for the Mk1 TreeScan system with good focus results. The internal scanner components were aligned to ensure no distortion during image capture.

Next a test was made to see whether real world dimensions could be estimated. This involved capturing images of a tall building used as a calibration object. The results of this test proved that the system was imprecise (see section 6.5).

Although imprecise, the system was operational so a field trial was undertaken to test the system under forest conditions. This was the first time the system was operated in conditions typical of a radiata pine forest. This field trial highlighted two points:

- The lighting conditions were unexpectedly low resulting in very long image capture times (timeout maximum of 20 minutes).
- It is very difficult to physically measure standing pine trees to the accuracy required to calibrate the TreeScan size estimates.

In response to the difficulty in measuring a tree as a calibration object, a 'metal tree' was built for system calibration. The 30 metre calibration tree lies horizontally and has been used to evaluate the system accuracy in two and three dimensions.

In an attempt to determine where the system imprecision lay, the perspective correction algorithm was redesigned. It was found that the two step correction method being used was an approximation, however, ^{this} did not explain the imprecision of the estimates. A different image correction method based on geometric calculations rather than planar transformation was implemented. By implementing three slightly different versions (see section 3.5.3) the cause of the imprecision was identified.

The calibration method being used was mathematically correct, but calibration rod dimensions were being used in calculations to a precision much greater than the measured precision. This resulted in random fluctuations in height estimates of approximately 6 meters at a height of 40 meters.

At this stage the Mk2 system was developed. The calibration procedure was modified so the angle of the camera with respect to the tree is physically measured, rather than derived from the calibration rod image dimensions.

A different lens was purchased to address the other two problems. The new lens had a larger aperture providing four times the light of the first lens. As no motorised wide aperture one inch format lenses were available a manual lens was purchased and a stepper motor fitted to drive the focus ring.

A third modification that could be made is to increase the system sensitivity by implementing a video preamplifier. This would amplify the video signal before the A/D converter. This is being developed as part of Mr Aaron Drysdale's masterate.

The Mk2 system was a new unit so all evaluation tests, including calibration and characterisation tests needed to be repeated. The Mk2 system was tested under forest conditions by imaging an entire MARVL plot. This provided information on image capture timing and image quality, as well as providing sample images for the automated parameter extraction research being undertaken by Dr Ralph Pugmire. The trees were not physically measured so calibration tests could not be completed. Two image features were identified: the images contained considerable banding due to problems in the integration time adjustment routine, and the images were not as sharp as expected.

The banding problem was easily corrected, but the source of the focus problem required further research undertaken as part of Mr Aaron Drysdale's masterate. The results showed the poor focus was primarily due to lens aberration at the wide aperture positions of the new lens.

Accuracy tests were completed in both two dimensions and three dimensions with very good results. Both height and width estimates were within the required specifications.

During this development cycle, software changes were continually made to; implement the modified algorithms, improve the functionality of the software, and allow images to be captured for specific experiments. Further testing on the use of the TreeScan system under forestry conditions is presently underway.

6.3 Hardware Calibration

The scanner hardware must be calibrated to ensure the scanner is mechanically capable of producing estimates to the required degree of accuracy.

The hardware calibration process consists of two tasks; the alignment of the scanner's internal components to avoid distortion, and the accurate measurement of the mirror step angle which is used in the parameter extraction software.

6.3.1 Scanner Component Alignment

Inside the scanner, the scanning mirror, the lens, and the CCD imaging sensor of the line scan camera must all be in alignment. If this is not the case several distortion effects will be introduced. Any of these distortions will have a significant effect on the dimension estimates taken from the images. There are four individual distortions that could be introduced or a combination of the four if more than one misalignment is present (see figure 6.1).

The CCD imaging sensor is permanently mounted inside the line scan camera and it is assumed this has been correctly factory aligned. The lens is mounted directly on to the line scan camera using a screw-on C lens mount and is also assumed to be correctly

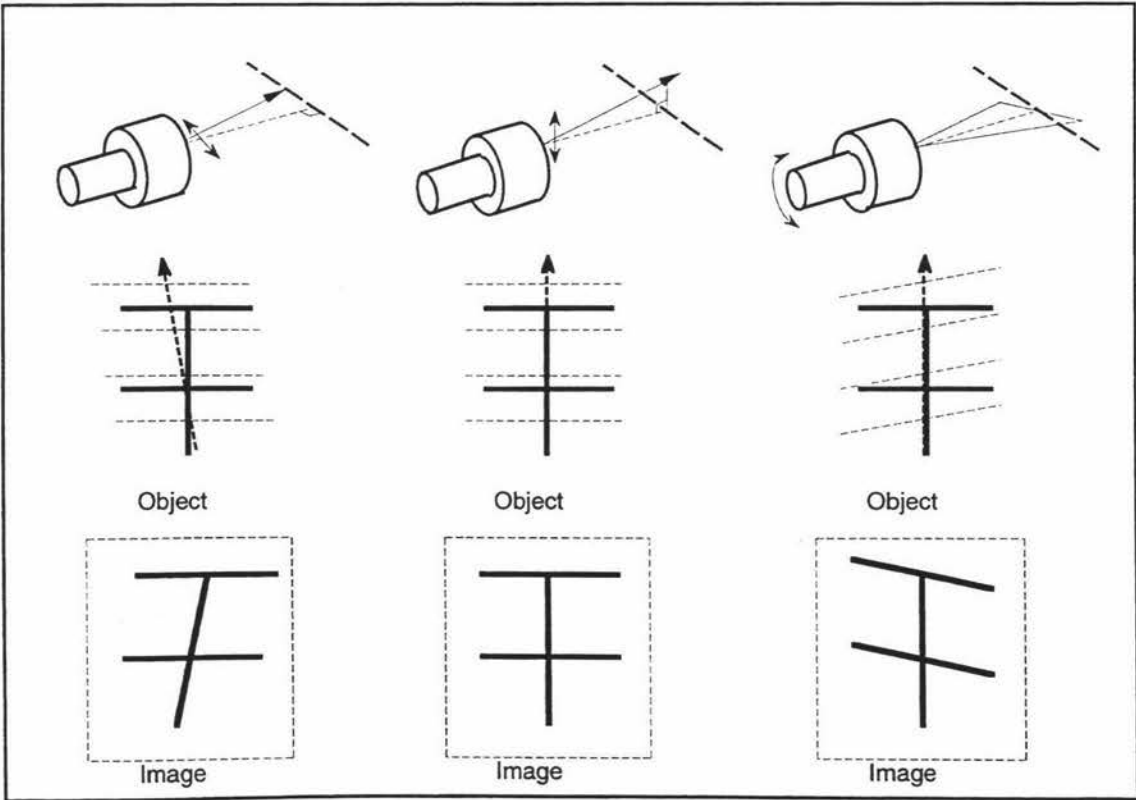


Figure 6.1 - Distortion introduced by camera misalignment

aligned. For the rest of this section the line scan camera and lens will be discussed as one unit and referred to as the 'camera'.

The scanning mirror is mounted on the precision rotation mechanism. This is permanently fixed to the scanner chassis and cannot be adjusted. The camera can be misaligned with the axis of the rotation mechanism in three ways and the mirror could be misaligned with the axis of the rotation mechanism. Each of these is discussed.

6.3.1.1 Camera Misalignment

The camera can be misaligned with the axis of the rotation mechanism in three ways as shown in figure 6.1:

1. The principal ray of the camera may not be perpendicular to the rotation axis. The principal ray scans sideways up the object and introduces a horizontal shear into the captured image.
2. The principal ray of the camera may be offset from the rotation axis. This offsets the position on the imaged object by the change in camera to mirror distance. This offset is very small and the effect of this misalignment is negligible.
3. The view angle of the camera may be at an angle to the rotation axis. This introduces a vertical shear into the captured image.

These misalignments must be corrected for. This can be completed by imaging a calibration grid and adjusting the position of the camera so that the camera and rotation axis are correctly aligned. By placing the scanner at 7.5 metres from a grid with lines 1.3 mm wide, it is ensured the grid lines are exactly one pixel wide.

If a captured image is distorted (see figure 6.2), this must be corrected for by aligning the camera in the horizontal plane to remove horizontal shear. If the image still contains a vertical shear the camera must be rotated until the image of the calibration grid is not distorted. Shims were used for fine adjustment of the camera position.

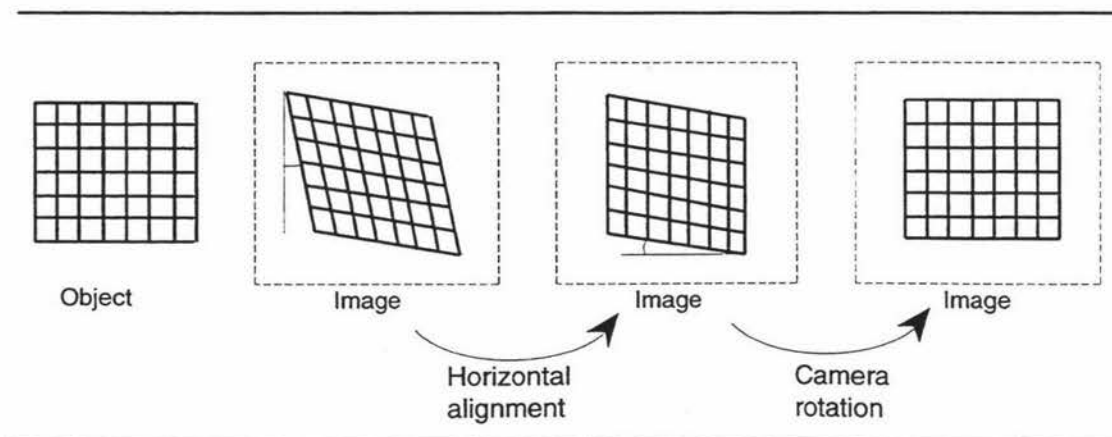


Figure 6.2 - Camera alignment procedure

6.3.1.2 Mirror Misalignment

The scanning mirror is mounted on the rotation mechanism. If there is a misalignment between the mirror and the axis of rotation of the mechanism, a nonlinear distortion will be introduced.

The nature of this nonlinear distortion depends on the angular position of the rotation mechanism over which the image is captured. The distortion introduces an 'apparent sweep' into the captured images so must be corrected for. The physical system was modelled in Matlab to determine the nature of this distortion, see appendix E for further detail.

The mirror was aligned to the axis of rotation of the mechanism to a within ± 0.01 degree.

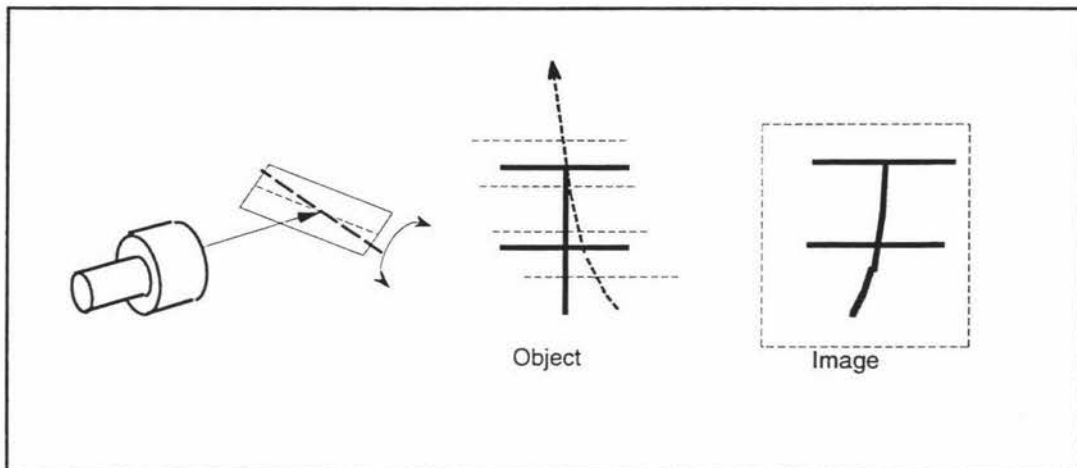


Figure 6.3 - Distortion introduced by mirror misalignment

6.3.2 Measurement of Step Angle

The precision rotation mechanism has been designed so that the step angle is approximately 0.01 degrees. The actual value of this angle, denoted alpha, is important as this determines the vertical pixel spacing. By using the number of pixels (steps) to a particular feature and the step angle, the real world feature dimensions are estimated using the perspective distortion correction discussed in chapter 3.

The step angle for the Mk1 system was measured to be $(1.030 \pm 0.002) \times 10^{-2}$ degrees, by measuring the number of steps required to complete a full 360 degree rotation of the rotation mechanism.

If the measured value of alpha is incorrect this will introduce a consistently high or consistently low error into estimated heights. The value of alpha was later empirically modified to 1.033×10^{-2} degrees, in order to correct for a consistently high error in height estimates during the two dimensional accuracy tests.

6.4 TreeScan Characterisation

The TreeScan characterisation is completed to determine scanner characteristics and establish the conditions associated with normal TreeScan operation. This provides important information on the aspects of the system that are satisfactory and those that may require further development work.

The TreeScan characterisation consists of a series of experiments to determine: the timing for a complete image capture cycle, the resolution to which features can be resolved, information on the integration time adjustment during image capture, and information on the performance of the autofocus algorithm.

6.4.1 Image Capture Timing

In this section the overall image capture timing is discussed. This section builds on the low level hardware timing discussed in section 5.2.3, and in section 6.1 where it was noted that the lighting under forest conditions was unexpectedly low.

The system typically takes 4 minutes to capture an image. This is the fastest possible image capture and may be longer in low light conditions. The time it takes to capture an image depends on two main factors:

1. Data conversion and transfer time
2. Integration time

Other factors that affect the image capture time are initial integration time adjustment, initial autofocus time, refocus time, integration time adjustment, and time required for additional processing tasks (see figure 6.4).

- In **normal day light conditions** the integration time is an order of magnitude smaller than the data conversion and transfer time. The scan time is limited by the data conversion and transfer time and the microcontroller is continually processing data.
- In **low light conditions**, such as those experienced in forests, however the situation is reversed. The integration time needs to be longer and limits the scan time. In this situation the microcontroller remains idle for extended periods.

The system must operate in a forest, and should operate as fast as possible. To speed up image capture in low light conditions modifications were made to the system to increase the light received by the imaging sensor and increase system sensitivity:

- An A/D conversion routine has been implemented that performs a 10 bit A/D conversion with an 8 bit lookup table. This increases the sensitivity by a factor of four by using the bottom quarter of the A/D dynamic range. However this is only a temporary measure as it requires more processing and extends the minimum scan time by 50% (see figures 5.9 and 6.4 for timing).
- A new lens provides approximately four times the light by having a larger aperture.
- A video amplifier is being built that will increase system sensitivity by amplifying the video signal before the A/D conversion.

The current timing of the Mk2 system is such that using the 10 bit A/D conversion the scan time is normally limited by the hardware limitation of data conversion and transfer time (4 minutes). With the video amplifier this is expected to be such that the image capture under forest conditions will not be limited by integration time.

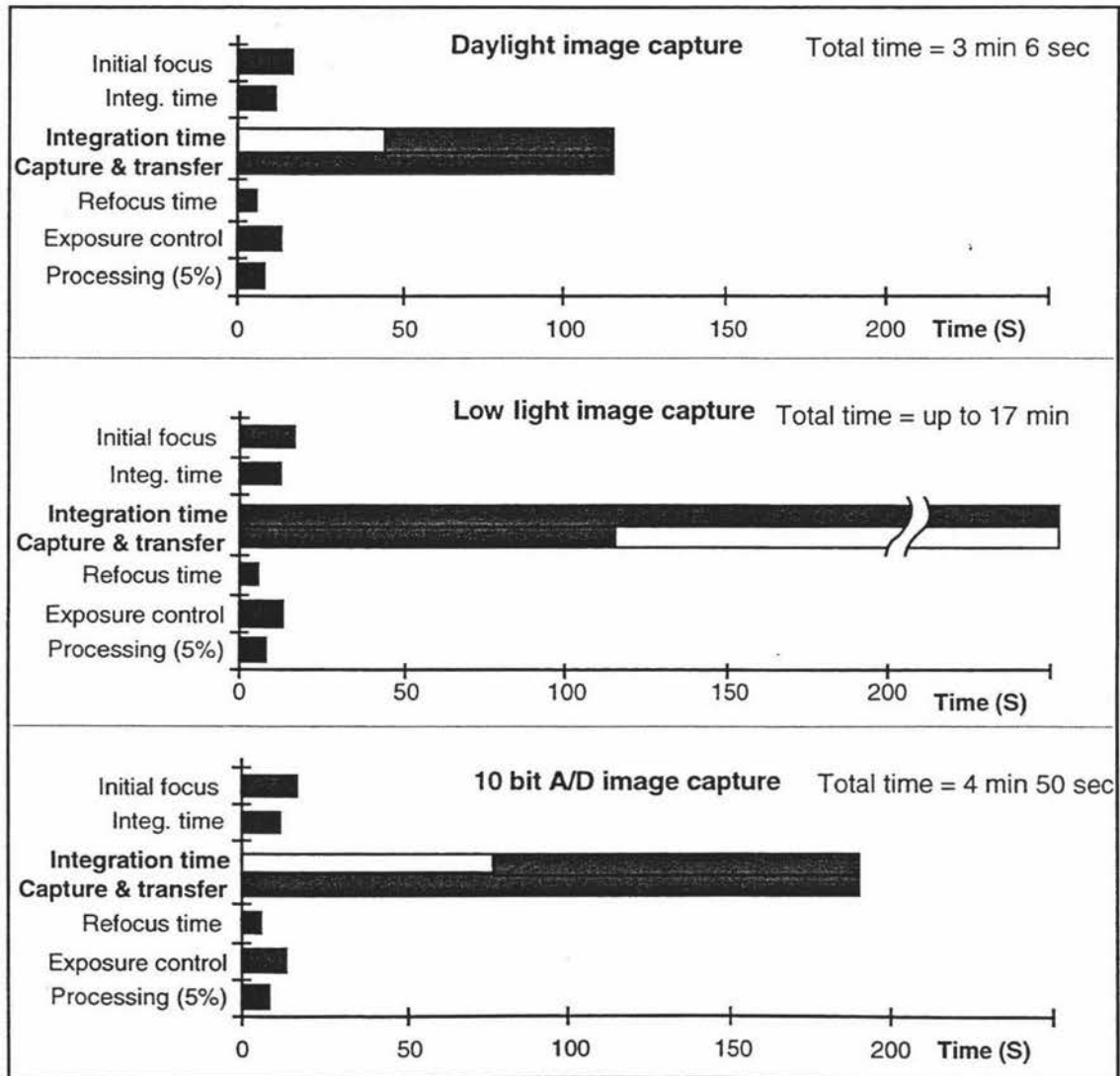


Figure 6.4 - Image capture timing

6.4.2 TreeScan Resolution

The resolution of the TreeScan system refers to the size of the smallest identifiable features on the TreeScan images. It is however important to distinguish between pixel resolution and discernible resolution; **pixel resolution** is the size a single image pixel represents on the real world object, **discernible resolution** is the resolution at which features of interest may be distinguished.

Pixel resolution and discernible resolution may be the same or different. In a situation where the image is sharp and the feature of interest spans several pixels the discernible resolution will be subpixel resolution. If however the image is out of focus or contains blooming the discernible resolution will be several pixels.

6.4.2.1 Pixel resolution

The pixel resolution is the size a single pixel represents on a real world object. The pixel resolution of the TreeScan system varies with position in the image; the pixel resolution at the base of the tree will be higher than the resolution near the top of the tree. The pixel resolution measured from the image matches the pixel resolution calculated in chapter three. The TreeScan pixels resolution for an image captured at 15 m from the calibration reference with zero dip and lean is shown in table 6.1.

Resolution	Height up tree		
	0 m	20 m	40 m
Horizontal resolution (cm / pixel)	0.27	0.42	0.72
Vertical resolution (cm / pixel)	0.27	0.68	2.2

Table 6.1 - Measured pixel resolution

6.4.2.2 Discernible resolution

The discernible resolution is the resolution at which features of interest can be distinguished. The discernible resolution depends on the quality of the image and the size and shape of the feature of interest. If an image is correctly focused and a feature spans several pixels it may be possible to determine the feature position to a sub pixel resolution. In a many situations however the image will not be perfectly focused or may suffer from defects such as blooming as a result of sensor saturation. In such a situation the image resolution will be less than the pixel resolution.

Examples of reduced discernible resolution are shown in figure 6.4. Around the stem in strongly backlit situations such as the top of the tree there may be considerable blooming, the stem may be obscured, or poor focus can result in blurred tree edges (see figure 6.1).

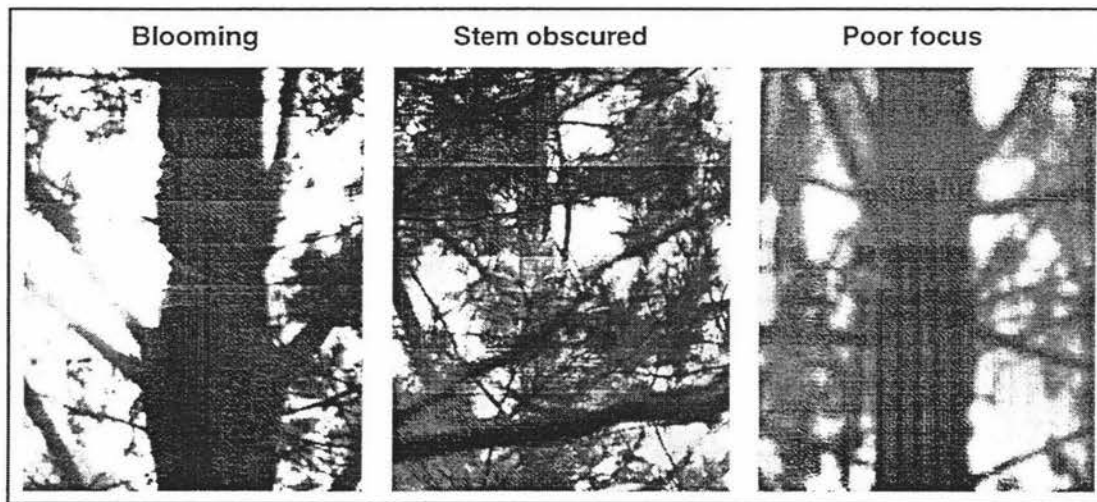


Figure 6.5 - Image resolution effects

6.4.3 Integration Time Adjustment

The integration time must be adjusted to prevent sensor saturation but provide enough light to allow adequate charge accumulation. If required, the integration time is adjusted throughout the scan.

During a typical image capture the lighting will be low with high contrast near the base of the tree. Near the top of the tree the lighting will be high contrast and backlit. Figure 6.5 shows the integration time adjustments for the capture of a 6000 line image.

The integration time is reduced by a factor of four near the top of the tree compared to the bottom. Also note that the integration time is not adjusted while scanning the calibration rod as the calibration circle would influence the integration time adjustments.

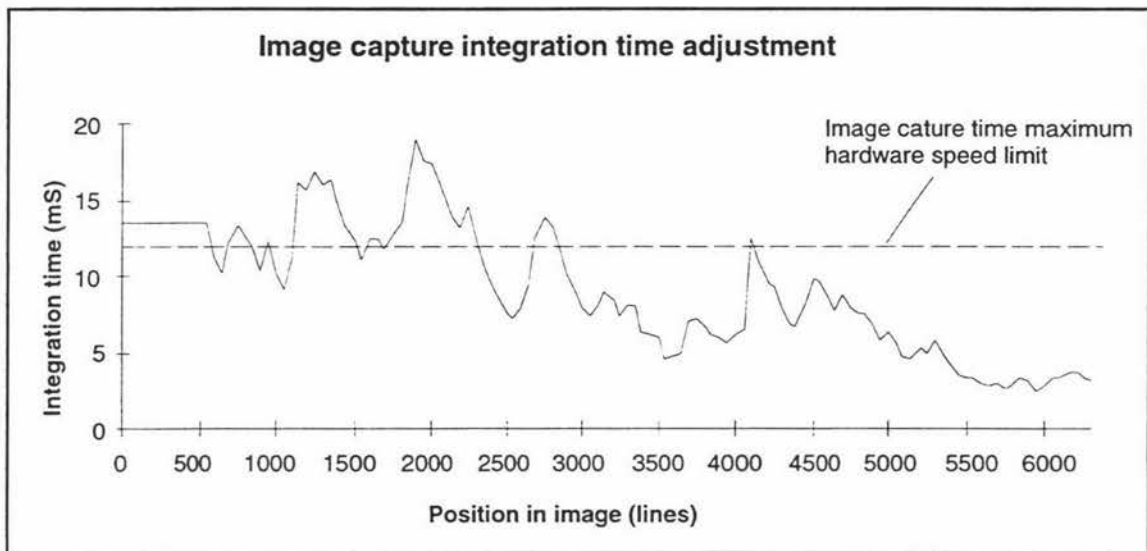


Figure 6.6 - Integration time adjustment

6.4.4 Focus Tests

The first verification that was completed was to characterise the focus measure under different operating conditions. It was found that the Brenner function worked well on both high and low contrast images. There was however variation in the focus measure as a result of variations in contrast, lighting, and the surface focused on.

The autofocus algorithm was developed based on a high contrast inside image capture situation. The results of the Mk1 system indicated that the focus algorithm worked very well on high and low contrast images both inside and out in the forest. The curve of the focus graph exhibited a sharp peak at the point of maximum focus. The curve was generally rising monotonically up to point of best focus then falling monotonically.

With the new lens of the Mk2 system however the results of the tests were not as conclusive. The focus curves sometimes contained significant noise and no clear peak at the point of optimal focus. In addition to this the images captured at the point of optimal focus were not as 'visually sharp' as the images captured with the Mk1 system. This required further characterisation and is currently under investigation as part of Mr Aaron Drysdale's masterate.

Figure 6.7 shows the focus trial under typical operating conditions. Variation within repeats of each measurement taken 0.1 seconds apart at the same mirror position are shown by the boxes and dashed markers on the graph. Also note that the searching around the best focus number in small steps has been vertically offset to improve clarity. See section 5.2.5 for a discussion on the autofocus algorithm.

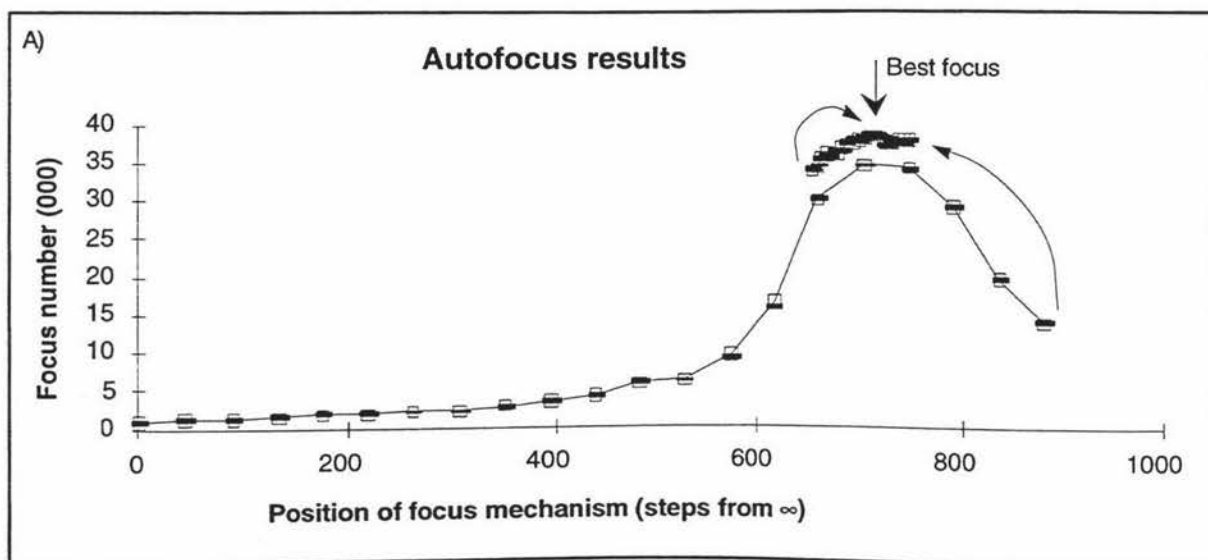


Figure 6.7 - Focus results

6.5 Initial Accuracy Tests in Two Dimensions

The aim of the accuracy tests is to validate that the TreeScan system is capable of providing sufficiently accurate and precise real world estimates. Initial testing restricted the dimensions to be estimated to the two dimensional calibration plane, reducing the possible sources of error by one degree of freedom. This section presents the results of the initial two dimensional accuracy tests which highlights the imprecision of the system when first developed.

The first parameters to be evaluated were height estimates on the calibration plane. These are the most prone to discrepancies and will highlight any accuracy or precision problems.

It is difficult to find a sufficiently tall (40 m) object that can be measured to within ± 1 cm required to calibrate the TreeScan system. A building with a regularly repeating pattern up its side was used. The building was measured using an accurate surveyors measuring tape.

The overall height estimates showed a very large error increasing with height. Height estimates produced systematically increasing or decreasing errors within single images, with magnitude and sign varying randomly within a group of images.

These two dimensional calibration tests proved that the Mk1 system did not have the required degree of precision and is further discussed in section 6.2. A detailed report can be found in Weehuizen and Pugmire (1994c).

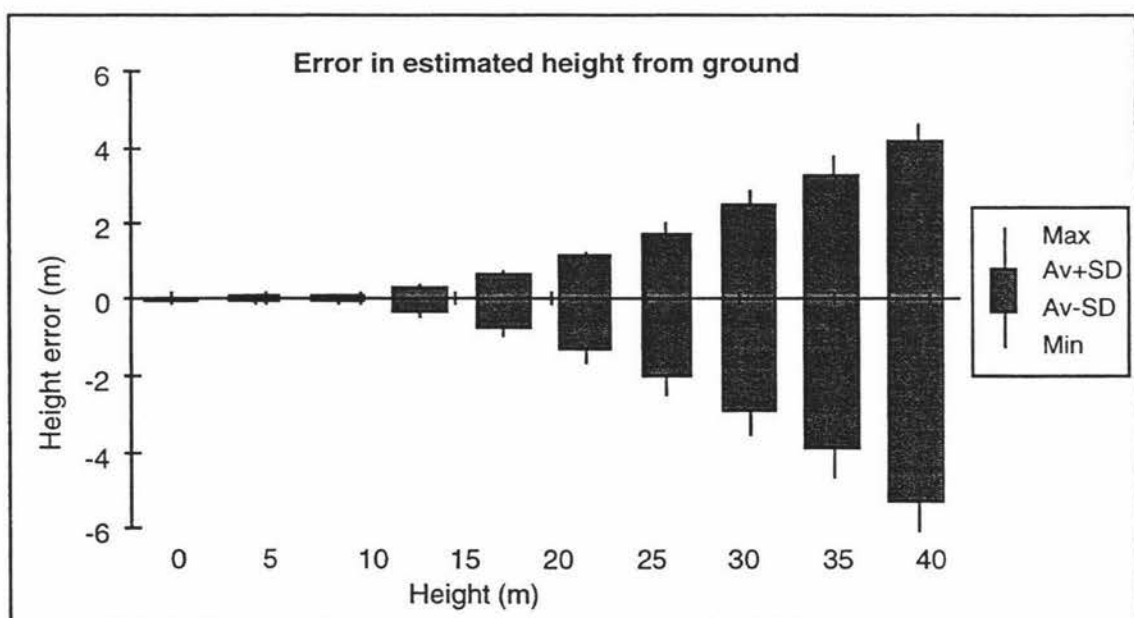


Figure 6.8 - Height errors with high imprecision

6.6 Final Accuracy Tests in Two Dimensions

Once the system had been modified and a more accurate calibration procedure established, further tests were complete to evaluate accuracy in two dimensions. These experiments validated that the TreeScan system is capable of providing sufficiently accurate and precise real world estimates.

Again the front face of a building with regular features was used as the calibration surface. Both height and diameter estimates were made; height estimates were made using the heights of the regular features up the building, and diameter estimates were made on 10 cm and 1 m horizontal features up the building.

A series of images was captured and estimates made from these. The resulting height estimates had a much greater precision, but contained a consistent offset of about 0.5 m at a height of 30 m. This was the result of imprecision in the measured value of α (see section 6.3.2). The angle α was empirically changed by 0.3% to correct the offset.

The final tests of the TreeScan system showed that:

- **Height estimates** in two dimensions can be estimated to a worst case precision of ± 20 cm at a height of 40 m (see figure 6.9a). There is an expected trend that errors in height estimates are larger near the top of the tree, with the result that the TreeScan precision is greater near the base of the tree.
- **Width estimates** in two dimensions can be estimated to a precision of at least ± 1 cm throughout the height of the tree (see figures 6.9b and 6.9c). The imprecision in width estimates is probably due to the manual placement of marking line endpoints to whole pixel accuracy.

These accuracy tests on TreeScan height and width estimates are based on twelve images captured from different positions and were completed by Mr Gary Allen. A more detailed analysis of the data can be found in Allan and Drysdale (1995a).

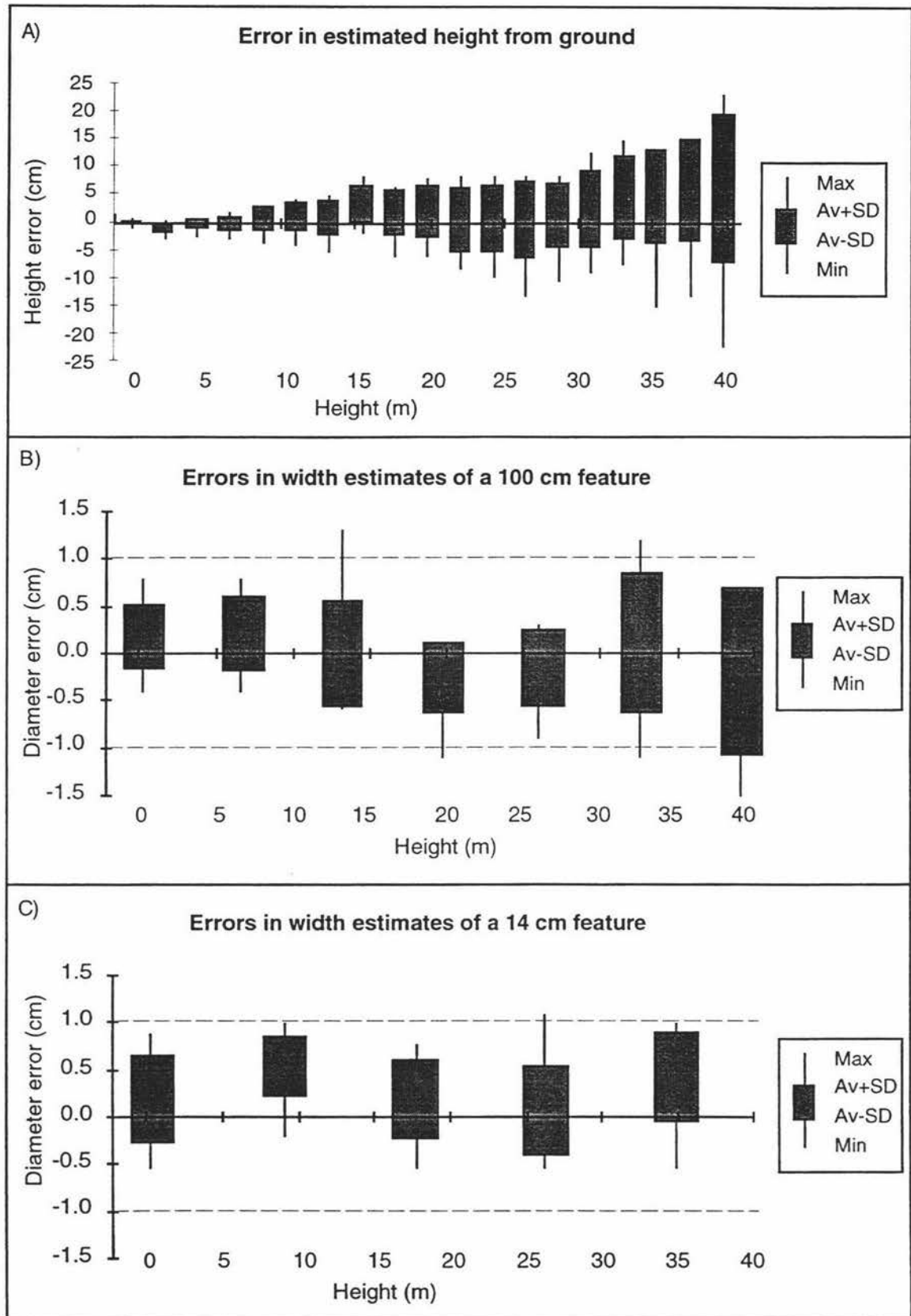


Figure 6.9 - Final accuracy tests in two dimensions

6.7 Accuracy Tests in Three Dimensions

The three dimensional accuracy test involve whether the 3D position of the tree stem and hence the tree shape can be accurately determined. Tests were completed on tree sweep estimates of the generated three dimensional model.

For these accuracy experiments it must be possible capture two orthogonal images of the calibration object. It must also be possible to modify the shape of the object in three dimensions to within ± 1 cm. To facilitate this a horizontal 'metal tree' built out of sheet metal was used with one image captured horizontally at ground level, and one image captured from above looking directly downwards.

It was found that although height and width estimates were good, an apparent sweep of approximately 6 cm was being introduced into sweep estimates. This was the result of slight mirror misalignment with the axis of rotation of the shaft introducing a slight curvature to the captured image (see section 6.3.1.2).

The final tests of the TreeScan system showed that:

- **Sweep estimates** in two dimensions can be estimated to a precision of ± 2 cm or typically one tenth stem diameter.

Further trials should be undertaken to fully characterise the three dimensional stem position estimates generated by the TreeScan system.

Chapter 7

FORESTRY IMPLICATIONS AND RECOMMENDATIONS

7.1	TreeScan Strengths and Limitations -----	162
7.2	Forestry Implications -----	166
7.3	Alternative Technology Uses -----	168
7.4	Future Work -----	169

The objective of this masterate research has been to develop a prototype line scan based computer imaging system to allow the dimensions of standing pinus radiata trees to be estimated. This has been successfully developed. However, the overall objective of this research has been to improve forest stand assessment by using imaging techniques to make the preharvest forest inventory information less subjective and more quantitative.

This chapter draws conclusions from the findings of the TreeScan evaluation trials and discusses the contribution the TreeScan system can make to in-field tree imaging. The role of the system in the forestry industry is discussed by highlighting strengths and limitations of the system. Lastly recommendations are made for alternative uses of this technology and future research on tree imaging.

7.1 TreeScan Strengths and Limitations

A prototype in-field tree imaging system has been built to estimate the dimensions of standing pine trees. The TreeScan system has been described at a logical and technical level, but if it is to succeed the TreeScan system will need to be used operationally in the forestry industry. At this level it is the usability and productivity that is achievable in a forestry environment that will determine the final role of such a system. In this section the TreeScan system is evaluated from a forestry viewpoint.

It should be remembered that the TreeScan system was developed as a "proof of concept" prototype and is the first generation of this technology. The main aim during the development was to develop a system that was physically capable of capturing images and providing accurate parameter estimates.

The TreeScan system has a number of strengths and limitations that will govern the eventual role of the system. The main strength of TreeScan is that it is the first system to provide an objective tool to characterise the three dimensional shape of a standing tree stem. However, the image capture takes four minutes and the two people are required to operate the system.

At this stage the use of the TreeScan system requires a trade-off between desired convenience and required accuracy of tree size information. For day to day operations it will remain more convenient to send a MARVL crew into the forest to visually estimate important tree parameters. The TreeScan system provides a research tool to gain more precise estimates of tree parameters, and provide calibration feedback on the accuracy of existing inventory methods.

TreeScan strengths	TreeScan limitations
<ul style="list-style-type: none"> Generates a 3D stem model (for sweep). Provides repeatable, accurate estimates of diameters and heights. Images can be processed in the forest. <p>Other strengths :</p> <ul style="list-style-type: none"> Little experience necessary. Can keep a visual record of the trees. 	<ul style="list-style-type: none"> Relatively slow image capture (~4 min) -> Possible tree movement. Semi manual processing of images. System requires two operators and careful setting up. <p>Other limitations :</p> <ul style="list-style-type: none"> System is sensitive so could get damaged or broken. Relatively high hardware costs. Large image data storage requirements.

Table 7.1 - TreeScan strengths and limitations

TreeScan strengths

The TreeScan system has three main strengths:

- The TreeScan system generates a model that characterises the shape of the entire tree stem in three dimensions. From this model objective estimates of sweep can be made. We believe this is the first device in the world to provide these facilities.
- In addition to sweep, the TreeScan system also allows other feature sizes to be estimated including; feature height, feature size, branch sizes, and stem diameter. Provided sufficient care is taken in image capture these estimates of tree size are very precise and accurate.
- The images captured can be processed in the field providing immediate feedback. This is very useful if the system were to be used to calibrate current MARVL crews.

Other strengths include;

- Estimation of tree parameters by the MARVL system requires considerable experience. This experience is not needed to operate the TreeScan system. A small amount of technical training is required to operate the TreeScan system.
- The system captures images of the trees which can provide a visual record of the trees that have been processed to be stored for future reference. This could be used to generate a forest description database with detailed tree information.

TreeScan limitations

The TreeScan system also has three main limitations:

- The image capture is relatively slow, typically 4 minutes per image, and in low light conditions could be longer. In addition to this the equipment needs to be set up for each scan, the computer started and the image saved. This could limit the productivity.

During this long scan time, the tops of trees may move in the wind, introducing a tree stem wobble in the image that is difficult to distinguish from wobble in the actual stem shape (see section 2.2.2). This is a direct result from the line scan approach adopted by the system.

- The processing of the images to get tree size estimates requires considerable processing. This is a tedious semi manual task with the operator marking all dimensions to be estimated. Research is underway to further automate this task.
- The system requires two operators and is rather bulky. The system has a combined weight of approximately 24 kg (Computer 5kg, scanner 8kg, tripod 4kg, scanner batteries 3kg, calibration rod 2kg) and is currently contained in two aluminium cases. The calibration rod needs to be carried separately. Backpack carrying cases should be investigated to make it easier to carry the system around the forest.

Other limitations of the system include;

- The system is sensitive and could get damaged or broken as a result of being carelessly operated in a forestry environment, for example the system could get dirt into the plugs or water into the electronics.
- The cost per TreeScan system is relatively high, each system costing approximately \$20000. This is considerably more than the equipment cost for a current MARVL crew, but this must be traded off against the unknown value of the extra information and precision it provides.
- The image data storage requirements are very high. If many images are going to be captured, some form of permanent large scale image archival system should be set up.

A system such as the TreeScan system will always involve some degree of uncertainty in the estimates of real world dimensions because extrapolation of the calibration geometry is used to determine the estimates of tree parameter dimensions. It is

important though to keep in mind the limitations of alternative systems, and the reasons this research was undertaken in the first place. Tree shape information is required to a greater accuracy than human estimates can provide using the MARVL system.

It is also important to distinguish between fundamental limitations and limitations of the implemented features. **Fundamental limitations** tend to be limitations of the hardware used or the geometry of the forestry situation. Fundamental limitations cannot be easily resolved or modified, for example, the distance from the tree at which images can be captured will typically need be in the range of 12 to 20 metres. This cannot be easily changed as other branches would obscure the tree stem being imaged.

Limitations of implemented features however, tend to be implemented software algorithms, which can easily be modified. For example the type of user interface or the sequence of steps in the image capture algorithm.

7.2 Forestry Implications

The TreeScan system was developed as a proof of concept prototype to evaluate the feasibility of using in-field tree imaging to improve the preharvest forest stand assessment. A system has been built and has been proven to be sufficiently accurate. It must now be evaluated in a holistic sense as a forestry tool. Possible roles of this technology in the forestry industry are discussed. Also discussed is the final role of the TreeScan system determined by its strengths and limitations. When evaluating the usefulness and potential of the TreeScan system it must be recognised that it is the first device of its kind in the world and that this is the first generation of this technology.

The next stage of the evaluation cycle is to compare the information generated by the TreeScan system with the results of other systems. TreeScan results should be compared to those produced by a skid site log optimisation system currently under development by Tasman Forestry. The results should also be compared with existing systems such as MARVL and AVIS. The final role of the system will depend on the outcomes of these tests and the direction chose by both the management at Tasman forestry and the research and development team at Massey University.

At this stage in-field tree imaging, and the TreeScan system in particular, could be envisaged in a variety of roles. The TreeScan system could be used as a tool for:

1. Research work only
2. Measuring individual trees used for MARVL training
3. Accurately measuring trees in research plots
4. An inventory replacement for MARVL
5. Measuring trees at skid sites prior to harvest
6. Making an inventory of every tree in the forest

The final three roles are very ambitious and given the current state of the technology, considerable improvement and development would need to be completed before the system is anywhere near capable of these tasks. However the system would be very useful to; (i) serve as an accurate measurement device in the assessment of sweep during MARVL crew training exercises, and (ii) to make accurate measurements of the trees in research plots.

During **MARVL training exercises** performed by the MARVL crews, trees are called by several crews then cut down and the actual dimensions compared. The

TreeScan system would provide an alternative method of measuring the tree while it is still standing.

A number of research plots are maintained under a variety of management regimes. These are called **permanent sample plots**. Trees within permanent sample plots are measured every year. This information is used for research into tree growth under different conditions, and the development of growth models such as taper functions for individual areas. The trees in these plots need to be accurately measured. This would be a very good application of the TreeScan system.

The TreeScan system could also provide advantages during **valuation** by providing quantitative evidence of the timber in a stand. This would also provide quantitative information for forest buyers.

Lastly it must be remembered that the TreeScan system is only one implementation of in-field tree imaging technology. Many other implementations are possible. The video system (VideoScan) discussed in chapter two (see section 2.2.1) currently is under development as a separate project by Mr Farshad Nourozi as a masterate project under the supervision of Prof. R.M. Hodgson and Dr. R.H. Pugmire. The Forestry Research Institute of New Zealand Ltd. are also working on an imaging system to capture tree sweep information.

7.3 Alternative Technology Uses

The system developed here consists of a specialist high resolution scanner and a series of programs that customise the system to measure pine trees. Alternative uses of this technology fall in two categories;

- The entire system could be used to measure large objects
- The scanner could be used to capture high resolution images for any application

The system developed here is a system designed for measuring tall or long two and three dimensional objects. In addition to the forestry industry the system could be used in a large number of other applications such as in the rapid measurement of buildings in the event of an earthquake. By rotating the scanner by 90 degrees a whole new dimension opens up and the system could be used to estimate parameters on any two dimensional horizontal surface, for example, position of boats over water or cars on a car park.

The second possibility for alternative applications is the use of just the scanner to capture images. The scanner developed is a specialised high resolution scanner with a very high aspect ratio. This could be useful for other applications where a high resolution image with a high aspect ratio is required. Situations where this might be applied is in panoramic imaging without wide angle lens distortion, or the use of imaging in orchards.

Applications will typically involve the imaging of objects that do not move around. If modifications are made to the system to allow faster image capture the system could even be used in applications that do require fast image capture.

Another unique feature of the scanner is that it provides imaging technology with a constant angular step size between pixels in the vertical direction. The angular step size of pixels changes with the position in the image for normal area cameras. This could be very useful for certain applications.

7.4 Future Work

In a complex system such as the TreeScan system, the system will never be finished and there will always be modifications and improvements that can be made. However at this stage the most important task still to be completed is to fit the TreeScan system as it stands into the inventory improvement framework.

Possible uses for the system within Tasman Forestry should be explored. It is important that highlighted uses for the system are realistic and provide genuinely useful information.

A series of trials should also be completed to compare the information generated by the TreeScan system with the information generated by other systems. The outcomes of these tests will determine the future direction of the system. This includes the deployment of the TreeScan system within Tasman Forestry Ltd. and possible commercialisation of the system.

Other areas where future work could be completed can be divided into three areas:

1. Ongoing research on TreeScan system improvements

Ongoing research on TreeScan system improvements includes research on a number of aspects of the system that have already been implemented but that could be improved. This includes research on automatic parameter extraction to replace the current semi manual method, hardware developments to reduce the image capture time, and other developments to make the system easier to operate and more manageable.

Image capture speed can be improved by the implementation of a video amplifier to increase the scanner sensitivity and the implementation of faster A/D technology such as dedicated A/D hardware or use of digital signal processing (DSP) technology.

The inclusion of in-built angle measurement sensors would make the system easier to operate, and backpack carrying cases would make the system more manageable.

2. Research on the use of alternative technologies

There will always be alternative technologies which can be implemented to develop an in-field tree imaging system.

The use of alternative technologies that may have advantages over the TreeScan system must remain an option. The VideoScan approach under development at Massey University as a separate project falls into this category.

Other methods to directly capture tree shape and dimensions that do not store images are also under consideration. A possible approach is the use of laser scanning techniques.

3. Alternative applications for TreeScan technology

Lastly, alternative applications for the TreeScan technology have been discussed in the previous section. Entire projects could be set up on any one of these alternative applications.

Chapter 8

SUMMARY

8.1	Summary	-----	172
-----	---------	-------	-----

8.1 Summary

Quality inventory information is essential for optimal resource utilisation in the forestry industry. The present MARVL system used for the preharvest inventory assessment in the forestry industry has a number of weaknesses. The MARVL system uses predominantly subjective assessment of tree parameters and has been developed to the point where it is limited by this subjectivity. This is particularly true in the assessment of sweep.

In-field tree imaging is a method which has been proposed to improve the preharvest inventory assessment of standing trees. It involves the application of digital imaging technology to this task. The method described generates a three dimensional model of each tree through the capture of two orthogonal images from ground level.

Three ways of implementing in-field tree imaging were identified as promising in an earlier feasibility study. The first of these has been developed to a proof of concept prototype. This fully operational prototype has been named the "**TreeScan**" system. This thesis describes the design, development, and evaluation of the TreeScan system.

The TreeScan system consists of a portable computer, a custom designed high resolution scanner with integral microcontroller, a calibration rod, and custom designed processing software. Images of the tree are captured directly into the portable computer using the scanner which contains a CCD line scan camera and a precision scanning mechanism. Captured images are analysed on the portable computer using customised image processing software to yield estimated real world tree dimensions and shape parameters. This involves a semi manual task where the operator identifies the dimensions to be estimated.

The TreeScan system provides quantitative estimates of five tree parameters; height, sweep, stem diameter, branch diameter, and feature separation such as internodal distance. In addition to these estimates, a three dimensional model is generated which can be further processed to determine the optimal stem breakdown into logs.

Design considerations

In the development of a "high tech" instrument such as the TreeScan system, it is very important to consider the design constraints and key technical aspects of the system before development as any one of a large number of considerations could limit the usefulness of the final system. These are discussed in chapters two and three.

Any system developed must be robust and capable of operating under normal forest conditions. These conditions include; difficult image capture geometry (see

section 3.7), poor lighting conditions, variable tree stocking, rugged and possibly steep terrain with undergrowth, tree movement in the wind, and outdoor weather conditions.

The system must be based on practical technology and capable of producing accurate results. Key technical aspects that need to be considered include; required resolution, correction of image capture distortion, choice of imaging sensor, computer to scanner interface design, image storage requirements, optical design, and tree parameter extraction methods.

TreeScan technical implementation

A technical description of the implementation of the TreeScan system is presented in the hardware and software chapters (chapters 4 and 5). Key aspects of the implementation are discussed in detail, these include:

- **The image capture system**

A CCD line scan camera based image capture system has been developed to capture the image data. The line scan camera places critical timing constraints on the rest of the TreeScan system, introducing a complicated timing interrelation between: image integration, analog to digital conversion of image data, and transfer of data to the computer.

- **The SCSI communications interface**

A SCSI interface was developed for the scanner to provide a high speed communications interface between the scanner and computer. This interface is used to send control commands to the scanner and transfer image data back to the computer.

- **Distortion correction and tree parameter extraction algorithms**

Distortion correction and parameter extraction algorithms have been developed to correct for perspective distortion introduced during image capture and to process the captured images to provide tree parameter estimates.

- **Software development**

To develop a system with the functionality of the TreeScan system, software needed to be implemented at four different levels; Macros, Pascal, C and assembler. To minimise development time, a strategy was adopted to implement all algorithms at the highest level possible.

System evaluation

The evaluation of a prototype such as the TreeScan system is a cyclical process of characterisation, calibration, and modification where necessary. During each cycle system knowledge is gained and modifications or improvements are made. Several limitations were identified in the Mk1 version the TreeScan system. A second prototype was built (Mk2 version) which largely overcame those limitations.

The Mk2 version of the TreeScan system has been fully characterised, calibrated and the accuracy of tree parameter estimation tested. The experiments performed confirmed that the TreeScan system is capable of providing sufficiently accurate and precise real world estimates. Height in two dimensions can generally be estimated to an accuracy of ± 20 cm. Stem and branch diameters are estimated to an accuracy of ± 1 cm, and tree sweep can normally be determined to an accuracy of ± 2 cm (or typically one tenth stem diameter).

The system developed has a number of strengths and limitations. The main strength is that it is the first imaging system in the world (we believe) to generate three dimensional models of standing trees and provide objective estimation of sweep. The main limitation of the system is that the image capture is slow (typically taking 4 minutes). This limits the productivity achievable with the system. There is also a danger that if the tree is moving in the wind an apparent stem wobble is introduced which is difficult to distinguish from tree shape deformation.

Future directions

The TreeScan system could be used in a variety of roles in the forestry industry, ranging from solely a research tool to a direct replacement for the current MARVL system. In the short term the most likely role for the TreeScan system is as a calibration device in the training of MARVL crews and as a method to measure trees within experimental research plots.

Further work is required to evaluate how the TreeScan system can be part of the inventory improvement framework and what the final role of the system should be. This includes an evaluation of the performance of the TreeScan system in comparison to existing inventory assessment methods, and investigation into the value of accurate inventory information on standing trees.

Other future work which should be undertaken is: improvement of the TreeScan system, research into the use of alternative technologies to implement in-field tree imaging, and research into alternative applications for the TreeScan technology.

REFERENCES

- Allen, G. and Drysdale, D. (1995a) : TreeScan Two Dimensional Accuracy Report, Department of Production Technology, Massey University, 1995.
- Allen, G. (1995b) : Operator Manual for the TreeScan System (version 1.0), Department of Production Technology, Massey University, 1995.
- ANSI (1986) : American National Standard for Information Systems - Small Computer Systems Interface (SCSI), American National Standards Institute, 1986.
- Deadman, M. W and Goulding, C. J. (1979) : A Method of Assessment of Recoverable Volume by Log Type, New Zealand Journal of Forestry Science 9(2):225-239, 1979.
- Forestry Facts & Figures (1994) : Forestry Facts & Figures, New Zealand Forest Owners Association Inc. in co-operation with The Ministry of Forestry, 1994.
- Groen, F.C.A., Young, I.T. and Ligthart, G. (1985) : A Comparison of Different Focus Functions for use in Autofocus Algorithms, Cytometry, Vol 6, 1985.
- Heavers, O.S. and Dichburn, R.W. (1991) : Insight into Optics, John Wiley and Sons Ltd., 1991.
- Horder, A. (1972) : The Manual of Photography, Focal Press Ltd., 1972.
- Jacobson, D. (1993) : Frequently Asked Questions Regarding Lenses, Electronic Internet Document : Lens FAQ, 1993.
- Jain, A. K. (1989) : Fundamentals of Digital Image Processing, Prentice Hall, 1989.
- Knoll, T. (1991) : Writing Plug-in Modules for Adobe Photoshop, Plug-in Developers Kit, Apple Software Developers CD.
- Loral Fairchild (1991) : Loral Fairchild 1991 CCD Imaging Databook, Loral Fairchild Imaging Sensors, 1991.

Protel Technology (1989) : Reference manual to Protel Schematic 3 and Protel Autotrax, Protel Technology Pty Ltd, 1989.

Pugmire, R. (1993) : Automation of Forest Stand Assessment Feasibility Study, Department of Production Technology, Massey University, 1993.

Pugmire, R. (1994) : An Experimental System for Forest Stand Assessment, Department of Production Technology, Massey University, 1994.

Rasband, W. (1993) : NIH Image User Manual, Wayne Rasband, 1993.

Ray, S. (1979) : The Photographic Lens, Focal Press Ltd., 1979.

Russ, J. C. (1992) : The Image Processing Handbook, CRC Press, 1992.

Schreiber, W. F. (1986) : Fundamentals of Electronic Imaging Systems, Massachusetts Institute of Technology, 1986.

Tasman (1995) : Technical Documentation for LOGOPT Log Optimiser, M. Ronnqvist and D. Ryan, Tasman Forestry Ltd., 1995.

Texas Instruments (1990) : SN75C091A SCSI Bus Controller Data Manual, Texas Instruments Inc, 1990.

Vivino, M. (1993) : Inside NIH Image Manual, Mark Vivino, 1993.

Weehuizen, M., Pugmire, R.H. (1994a): The use of In-field Tree Imaging in the Pre-harvest Inventory Assessment in the Logging Industry, Proceedings of New Zealand Postgraduate Conference for Engineering and Technology Students, Department of Production Technology, Massey University, 1994.

Weehuizen, M., Pugmire, R.H. (1994b): The use of In-field Tree Imaging in the Pre-harvest Inventory Assessment in the Logging Industry, Proceedings of the Second New Zealand conference on Image Vision and Computing, Department of Production Technology, Massey University, 1994.

Weehuizen, M. and Pugmire, R.H. (1994c) : TreeScan Characterisation and Calibration Report, Department of Production Technology, Massey University, 1994.

Weehuizen, M. (1995) : Technical Reference Manual for the TreeScan System (Volume 1 - Main Manual, Volume 2 - Software Manual), Department of Production Technology, Massey University, 1995.

Wolf, P. R. (1974) : Elements of Photogrammetry (With Air Photo Interpretation and Remote Sensing), McGraw Hill, 1974.

Yeo, T.T.E., Ong, S.H., Jayasooriah and Sinniah, R. (1993) : Autofocussing for Tissue Microscopy, Image and Vision Computing, Volume 11 Number 10, 1993.

Appendix A

Development Documentation for the TreeScan System

Reports Produced by Production Technology

- Automation of Forest Stand Assessment Feasibility Study, R.H. Pugmire, December 1993. (24 Pages + Appendices)
- An Experimental System for Forest Stand Assessment, R.H. Pugmire, January 1994. (7 Pages + Appendices)
- Brief outline of the Measurement Deviation of the Experimental Image Capture System from Actual Measured Parameters, M. Weehuizen, May 1994.
(2 Pages + Appendices)
- TreeScan Characterisation and Calibration Report, M. Weehuizen & R.H. Pugmire, December 1994. (34 Pages)
- TreeScan Two Dimensional Accuracy, G. Allen & A. Drysdale, August 1995.
(8 Pages + Appendices)
- Operator Manual for the TreeScan System (version 1.0), G. Allen, January 1995.
(33 Pages)
- Technical Reference Manual for the TreeScan System, M. Weehuizen, September 1995.
 - Volume 1 - Main Manual. (50 Pages + Appendices)
 - Volume 2 - Software Listings. (140 Pages)

Reports Produced by Tasman Forestry

- Tree Imaging Project : Background Notes and Specifications, M. Colley, December 1993. (4 Pages)

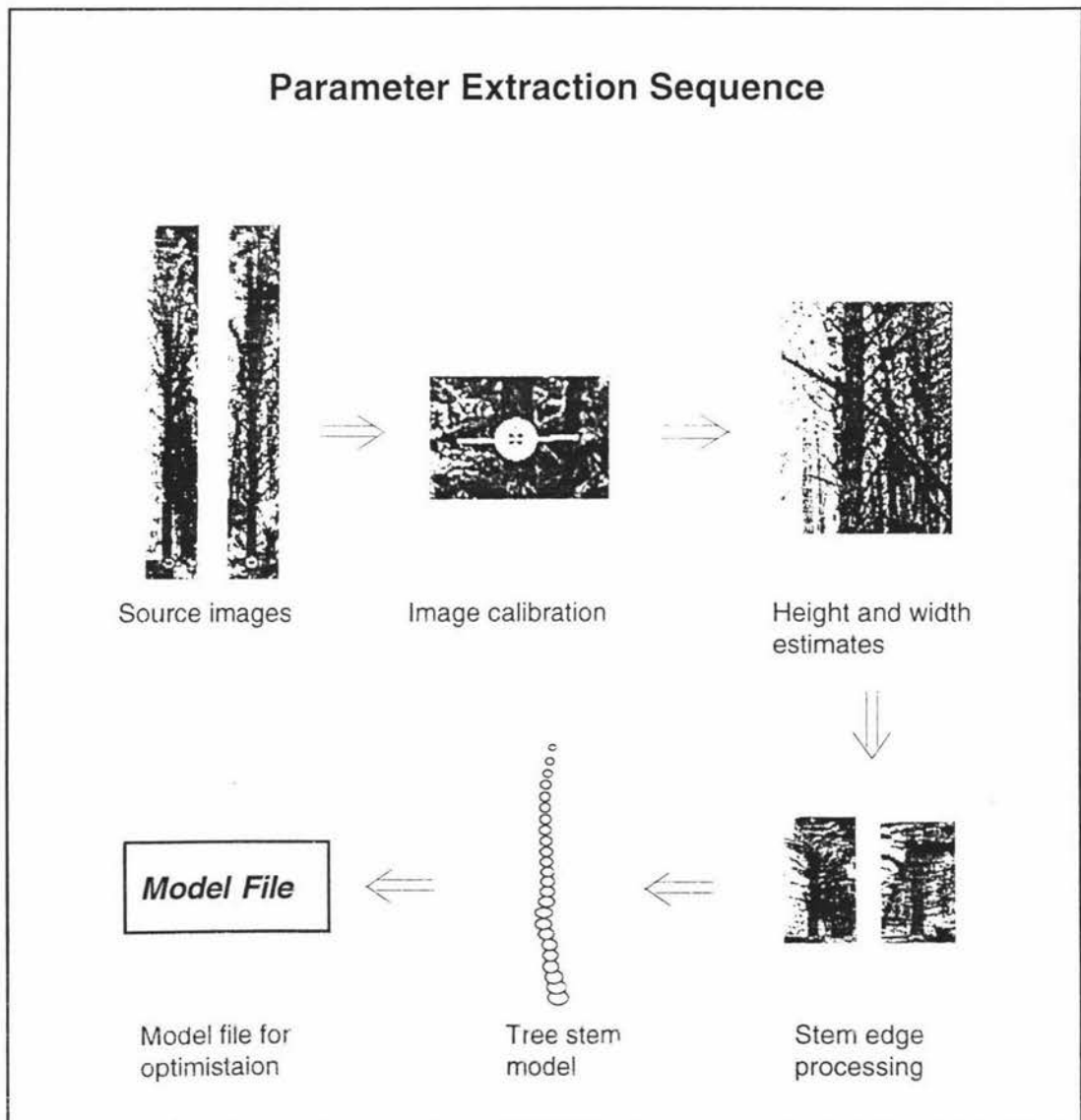
Project Proposals Produced by Production Technology

- Sub-Project 1 Proposal : Line Scan Camera Image Capture Prototype,
R.H. Pugmire, December 1993. (4 Pages)

Appendix B

Sample Tree Analysis

This appendix presents a sample set of images of one tree and follows these images through the processing stages to extract the tree parameters. The tree has a fork near the top of the tree.



Source images at 25 % full scale

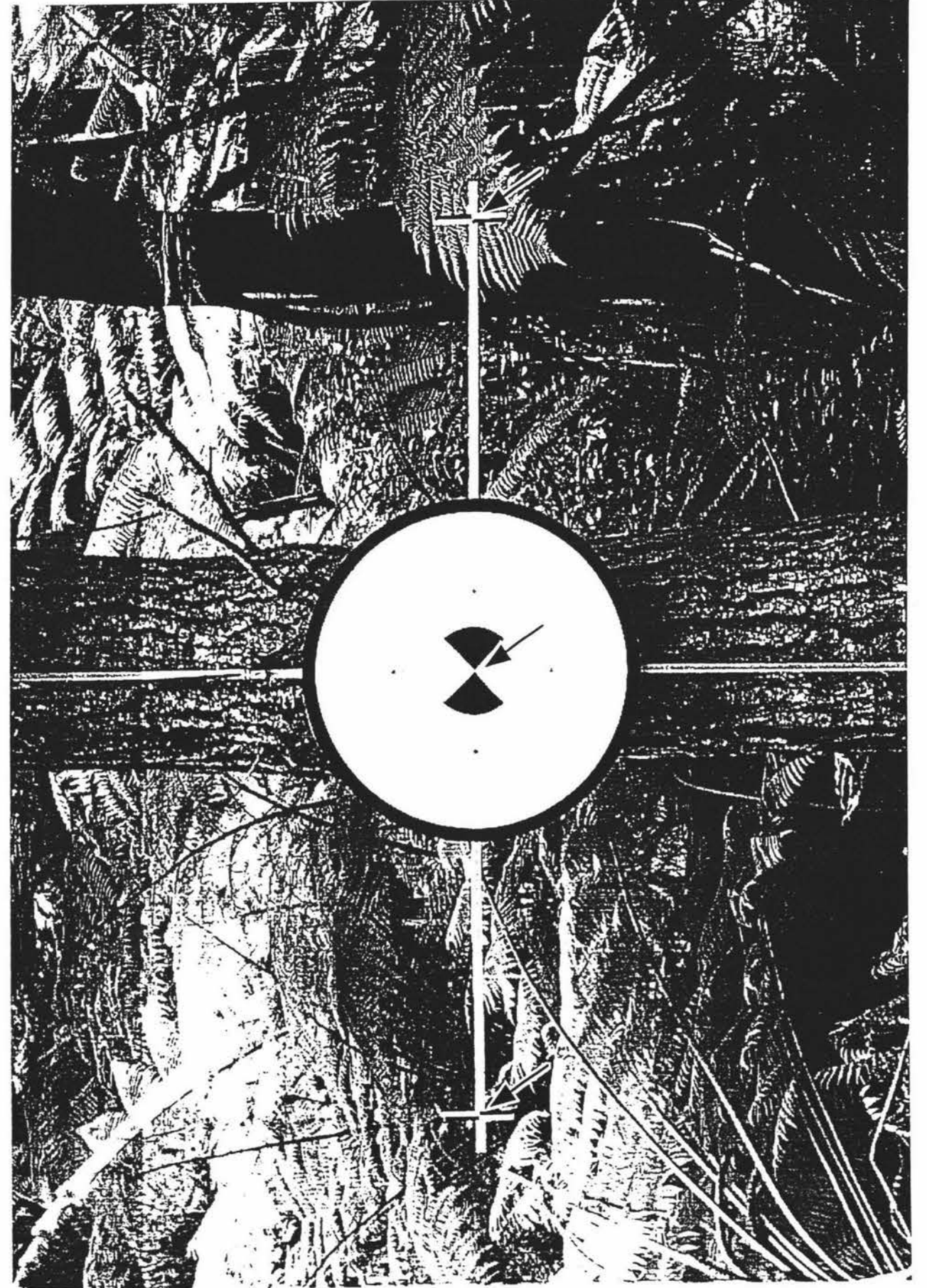
View 1 (6927x1024 pixels)

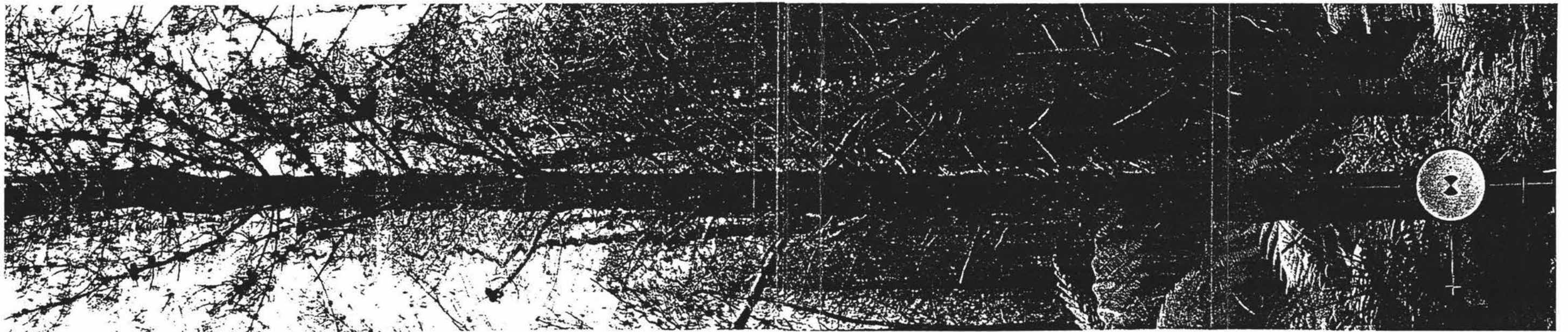
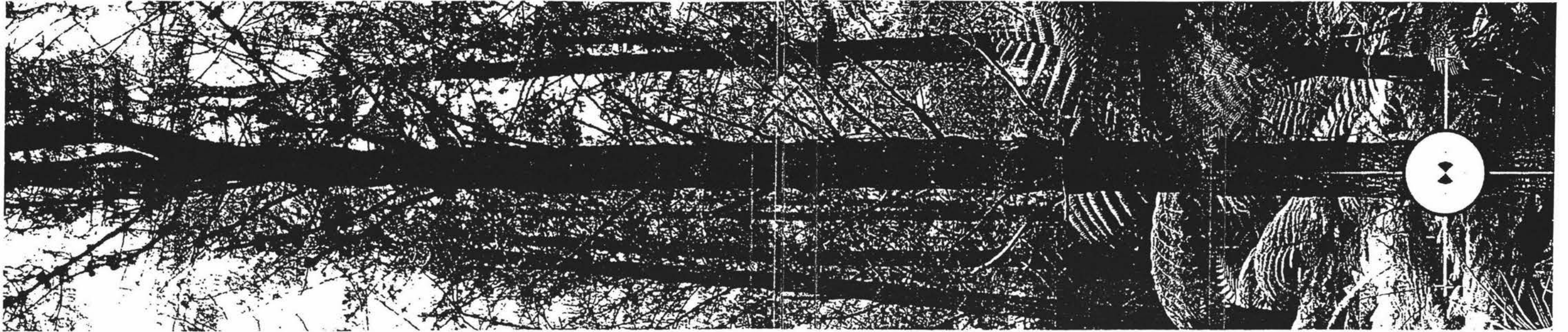


View 2 (6755x1024 pixels)



Marking of calibration points





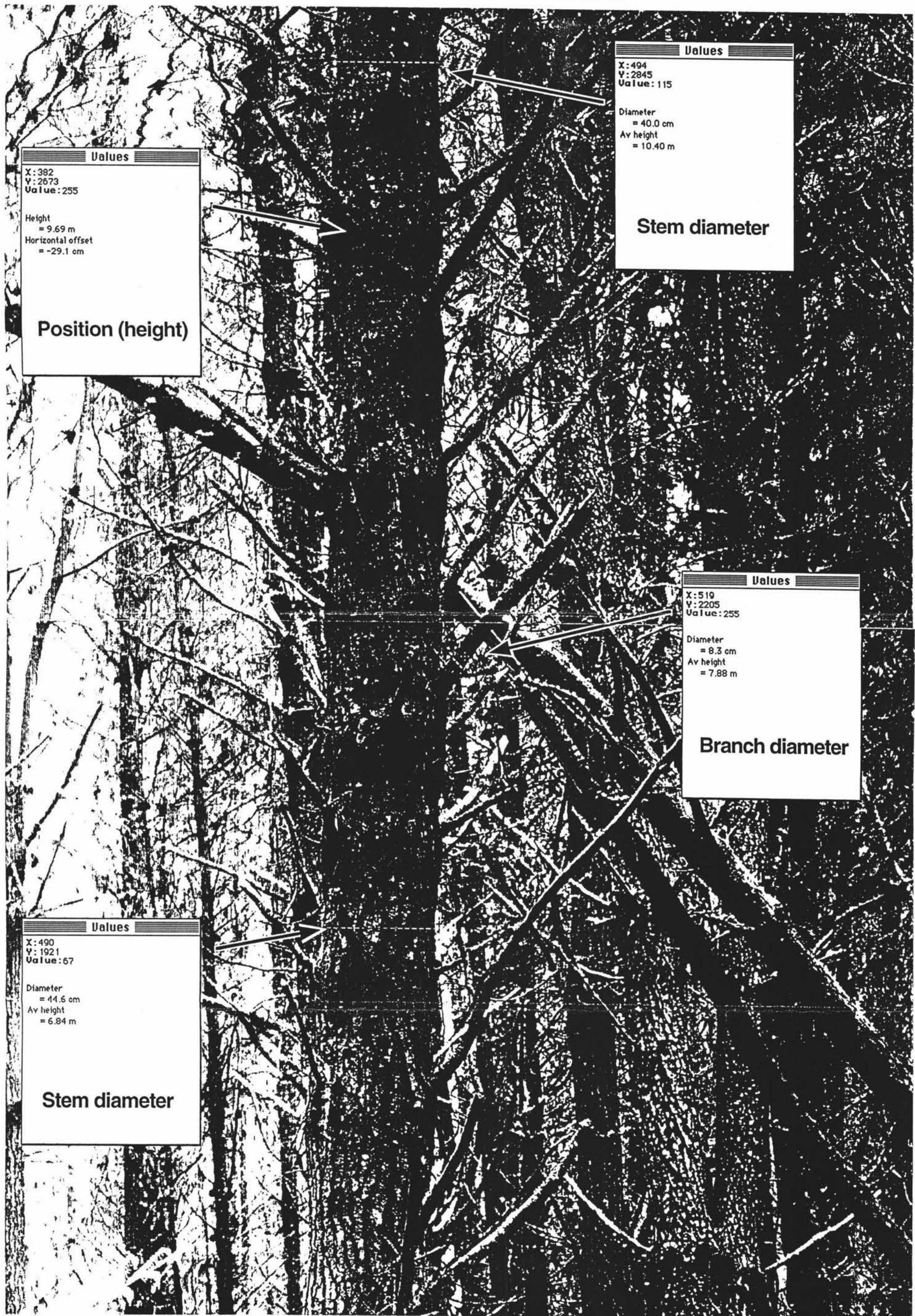
**Source images vertically decimated by 4 and lines marked
for 3D model generation at 30 % full scale**



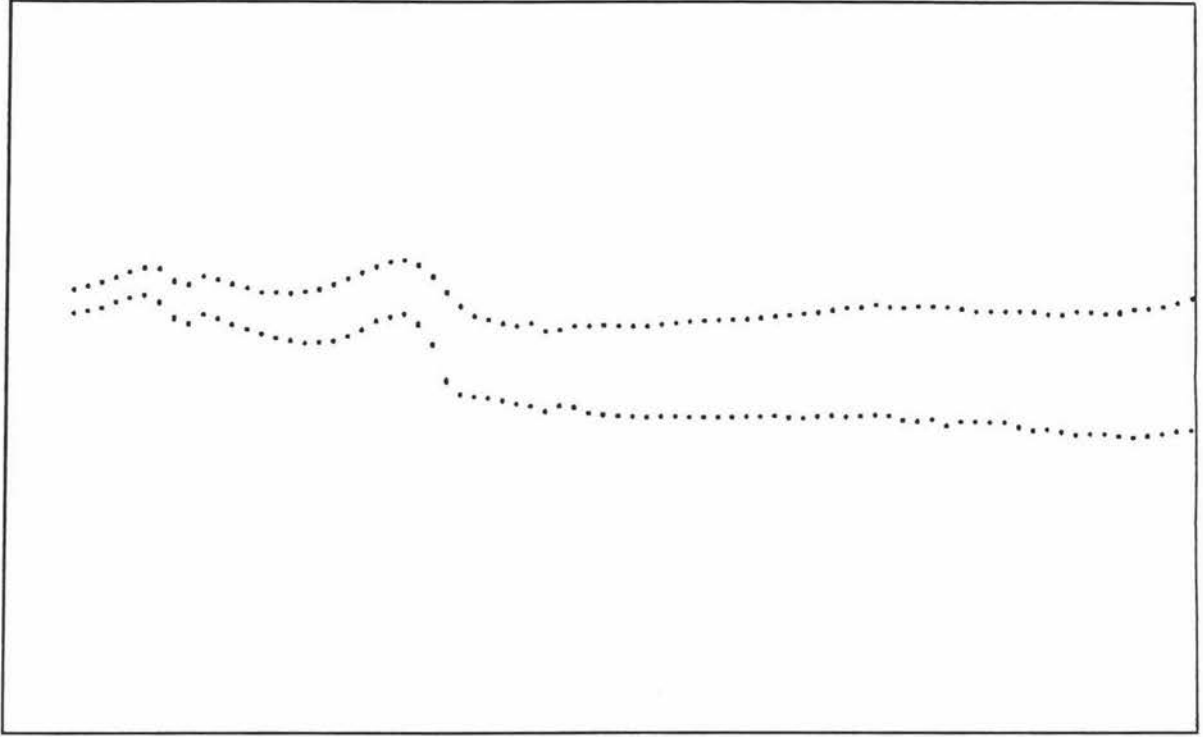
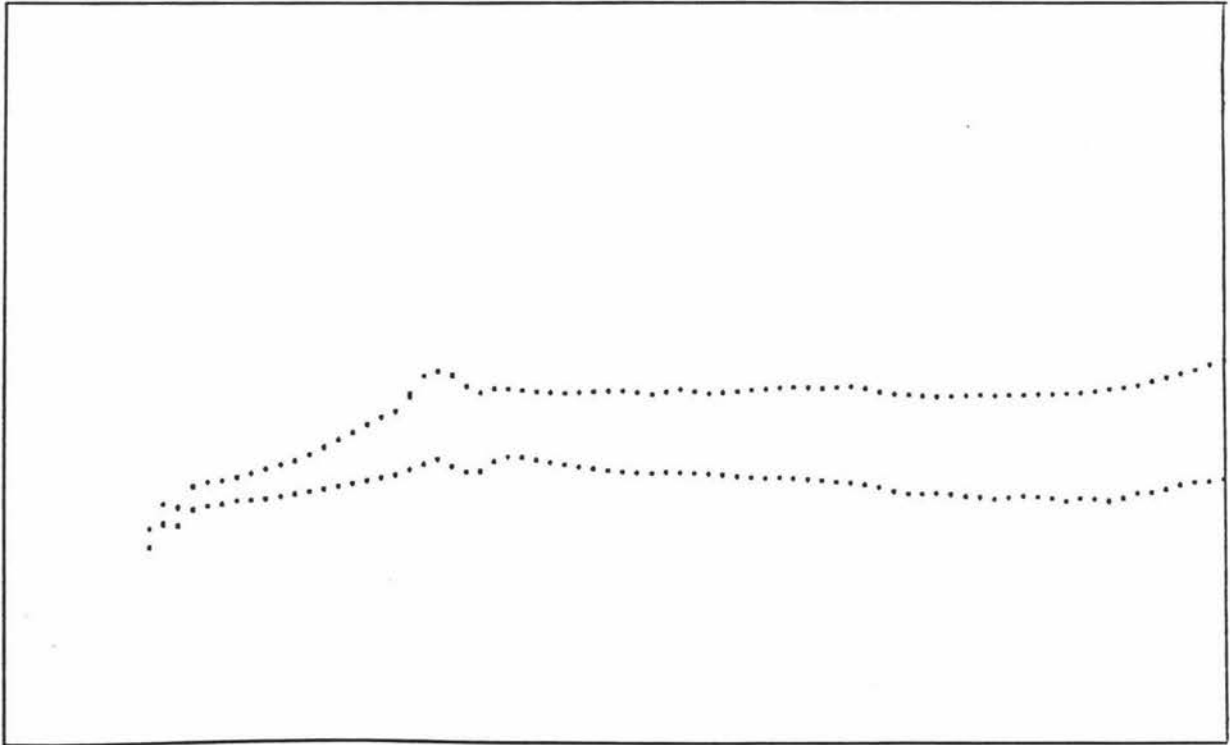
(line width increased to visually improve the image)



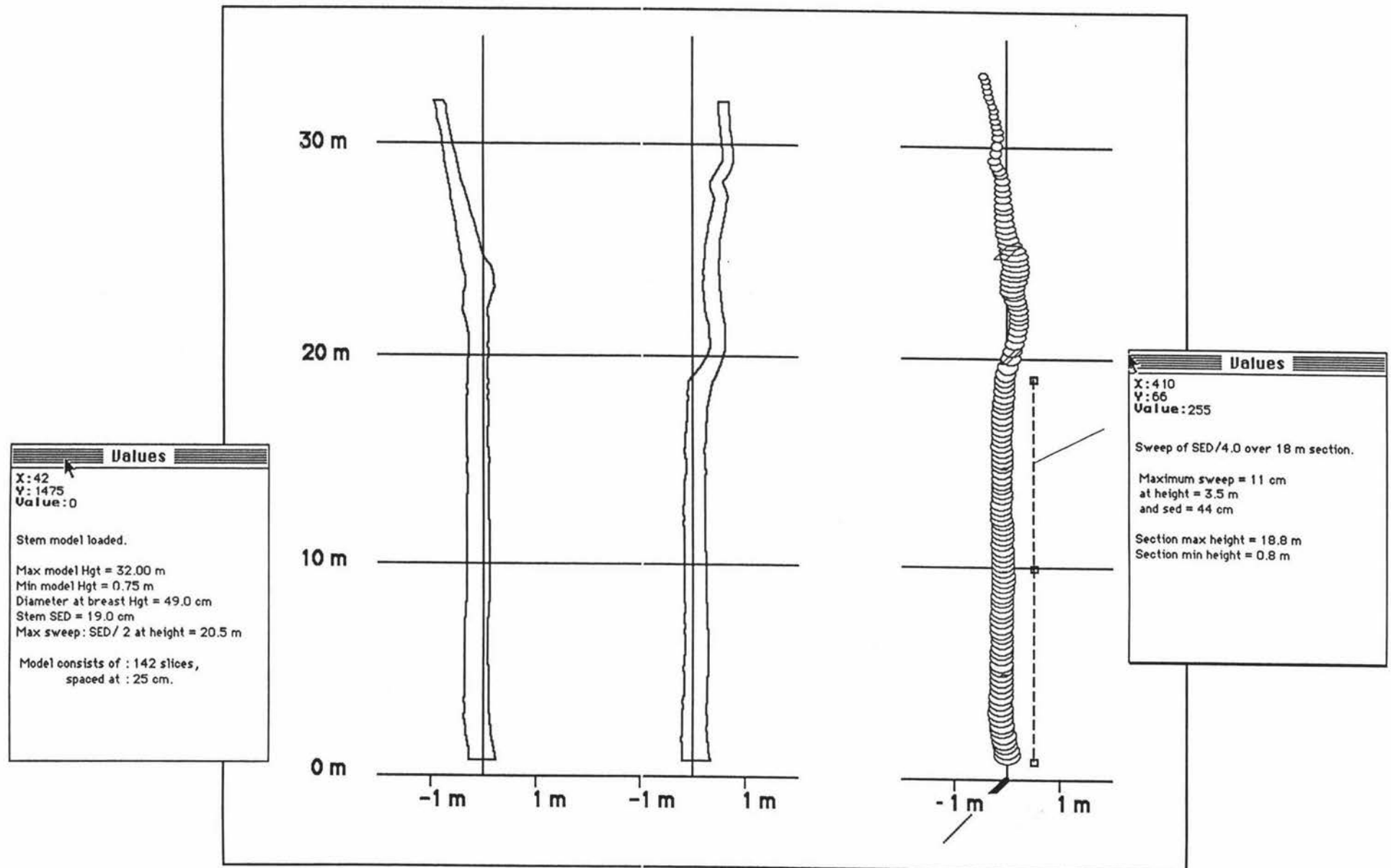
Tree parameter estimation in two dimensions



Stem edge points
(point size increased and frequency reduced
to visually improve the image)



Display of stem model allowing sweep estimates to be made



Appendix C

Forestry Terms

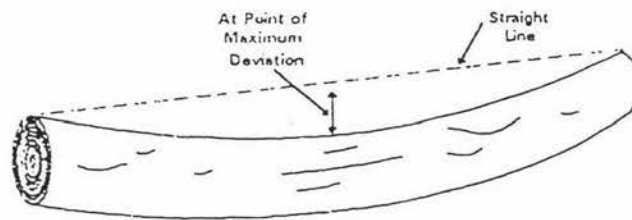
C.1 Definitions

These are definitions of forestry terms as defined for development of the log optimisation software developed by Tasman Forestry Ltd.

Sweep

The maximum deviation from straightness along the length of the log.

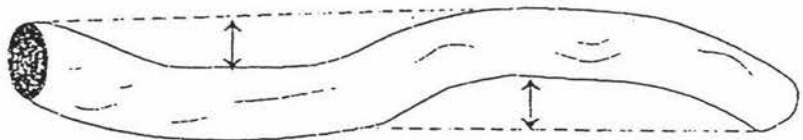
- Sweep shall be specified as D/x over the log length.
- Where D is the average diameter at the point of maximum deflection and x is the magnitude of deflection.
- Maximum deflection shall be measured where the tape is stretched from the middle of one end of the log to the middle of the other end of the log.



Note : Sweep has previously been defined as $\frac{\text{maximum deflection}}{\text{log small end diameter}}$. The definition of sweep based on log SED is still used in most other forestry applications.

Wobble (cm)

A defect where the axis of the log deviates in 2 or more different directions along its length. (To a maximum of 6 meters.) The bends in the log may be in the same plane, at right angles to each other or from a log which spirals.



- Wobble is the larger of the two deflections shown, not their sum.
- Wobble is expressed either as an absolute amount or as a ratio of the maximum deflection to the average log diameter at point of maximum deflection.

Kink (cm)

A short deflection in the log affecting less than 2m of the log.



- Kink is measured as the maximum deviation of the axis

Diameters**Small End Diameter (SED):**

- Minimum Diameter
(Of two diameter measurements at right angles through pith.)
- Measured under bark.

Large End Diameter (LED):

- Maximum diameter anywhere on the log.
(Usually, but not necessarily, located at the large end.)
- Measured under bark.

Diameter Breast Height (DBH):

- The average diameter of the tree measured at a point 1.4m above ground.
- DBH is expressed as an 'over bark' measurement
- For Log Optimisation purposes breast height is 1.4m minus the stump height from the butt. The average over bark diameter may be extrapolated from measurements taken above and below this point.

Internode**Internode Minimum**

The minimum clear distance (in mm) between whorls on a log.

Internode Maximum

The maximum clear distance (in mm) between whorls on a log.

C.2 Log Grades

On the skid site tree stems are cut into logs of a certain grade. The log grade is a measure of quality and value of each log. Each log grade has specifications which a log must meet. Tasman Forestry harvests over 50 different log grades.

Log grade specifications are based on:

- **Length** - Minimum length, maximum length, and average length with a standard error are specified for each log grade.
- **Diameter** - Minimum, maximum and average values are specified for SED, LED, and average diameter are specified for each log grade.
- **Shape** - Log shape restrictions specified by maximum sweep, wobble and kink allowances are specified for each log grade.
- **Knots** - Maximum knot size and knot frequency are important specifications for each log grade.
- **Features and defects** - Other features and defects such as rot, nodal swelling, and fluting are important specifications for each log grade.

For example, a summarised specification of **Japanese A Grade** logs consists of:

Minimum Lengths :	4.10 m	
	8.10 m	
	12.10 m	
Diameters :	Minimum SED	20 cm, or as directed.
	Minimum average SED	33 cm.
	Maximum SED	70 cm.
Knots :	Maximum 15 cm or 1/3 SED, including collar.	
	Maximum spike knot 8 cm or 1/4 SED.	
Wobble :	Up to 5 cm wobble is permitted.	
Kink :	Not permitted.	
Roundness :	No restriction.	
Pith :	No restriction.	
Fluting :	No restriction.	

Bark Damage :	Bark damage resulting in discolouration, decay, trimming flush with barrel of tree or cutface is not permitted
General :	No draw-wood. No rot, stain, or drywood. No splits. No saw cuts. No machine damage. Ends cut square.
Marking :	Every log to have a Green "A", Tasman logo, and crew no. in green on large end.

For inventory management purposes five standard log grades are defined by Tasman Forestry:

- **pruned**
- **Japan A**
- **Korea K**
- **domestic sawlog**
- **pulp**

These are used when using the MARVL data to predict possible harvests for individual stands.

Appendix D

Original TreeScan Project Proposal

The original TreeScan project proposal produced by the Department of Production Technology and the original background notes and specifications produced by Tasman Forestry are presented in this appendix:

- Line Scan Camera Image Capture Prototype : Sub-Project 1 Proposal
- Tree Imaging Project : Background Notes and Specifications

Line Scan Camera Image Capture Prototype

Sub-Project 1 Proposal

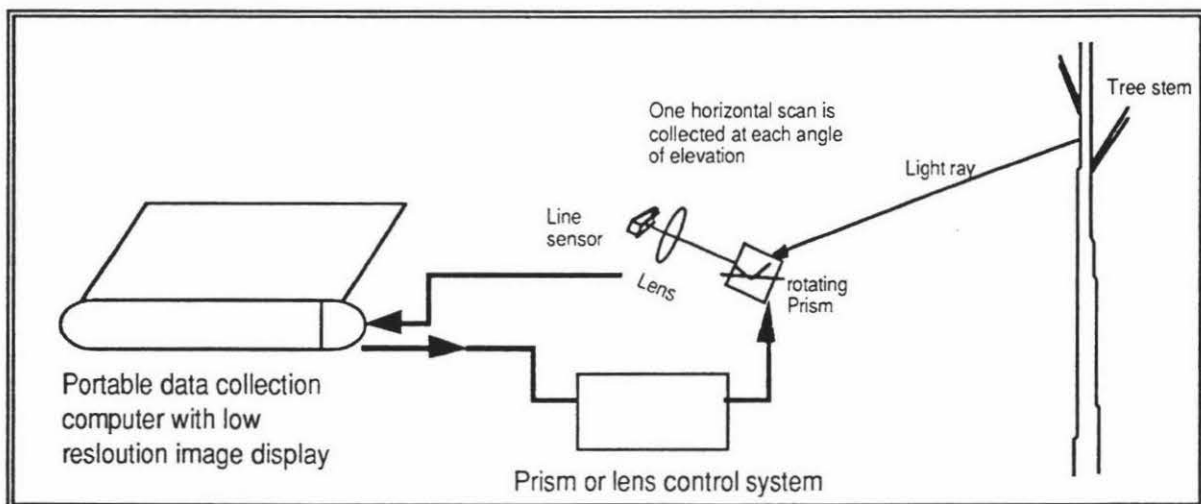
Project Outline

Development of a prototype line scan based image capture system for automation of forest stand assessment.

The proposed image capture system is outlined in section 7.3 of the feasibility study prepared for Tasman Forestry by the Department of Production Technology. The system is intended to capture images which can then be used to determine important tree parameters including sweep, diameter of stem and branch size. The main components of the system are shown in the diagram below.

This project is intended to produce a working prototype which will allow capture of images of trees and transfer to a portable computer. The captured images would be in a form that could initially be analysed using the NIH Image package and macro's produced as part of the feasibility study. The intention is to produce a system with a resolution of approximately 8000×200 which for a 40 metre tree will translate into a resolution of 0.5cm at breast height.

The intention is to produce a basic working system quickly, to which later refinements can be added. One such refinement could be the use of synchronised or structured lighting to improve contrast and aid in automatic extraction of tree parameters.



Proposed System

Format of project

The project is to be handled as a funded masterate project supervised by Ralph Pugmire and Professor Bob Hodgson. In order to speed development of the prototype technical support of approximately 1/3 of a man year will be

made available to the project from within the Department of Production Technology. The masterate student and technical support personnel will be chosen by the department.

Timing:

The intention is to provide a prototype system by the 1st of July 1993. The masterate is expected to be completed within 12 months.

Personnel

The present project team is listed below although this may change during the course of the project. In particular technical support may be provided by a number of people within the Department and other academic staff with particular areas of expertise will be involved in parts of the project.

Ralph Pugmire	Project Leader, supervisor for masterate
Marijn Weehuizen	Masterate student
Prof Bob Hodgson	First supervisor for masterate,
Farshad Nourozi	Technical support

Project monitoring

Brief one page reports will be provided on the status of the project monthly. A meeting with Tasman personnel should be held at least once each three months. We would also recommend that a Vis-A-Vis system be purchased by Tasman to enable the project team to liaise with Tasman during the course of the project.

Publication

- Any publication of aspects of this work must be cleared by Tasman prior to publication. Tasman may require a delay in publication of up to twelve months if the publication contains commercially sensitive material.

Naturally where possible we would like to publish the novel aspects of the work sooner than this and Tasman will endeavour not be overly restrictive in this area.

TASMAN FORESTRY LIMITED

TREE IMAGING PROJECT

BACKGROUND NOTES AND SPECIFICATIONS

Background

Tasman Forestry routinely assesses stands of mature radiata pine one to two years prior to harvest. This pre-harvest inventory typically involves the establishment of circular 0.04ha to 0.06ha plots on a systematic grid throughout each stand. The current objective is to obtain an estimate of stand recoverable volume per hectare with a PLE (Probable Limit of Error) not exceeding +/- 10% of the mean. In practice this requires a 2% to 4% sample by area of a typical stand of 20-40 hectares.

The pre-harvest assessment involves measuring diameters at breast height of all trees on a plot, measuring heights of a subset of trees, and an ocular assessment of features along the length of each tree in the plot. Features include branch size class, sweep (i.e. sinuosity) class, forks and broken tops. Information thus gathered is input to a computer which then proceeds to estimate the volume of each tree and the breakdown of that volume into various specified log grades, using an optimisation procedure that ensures the mix of log grades that will maximise value per tree is cut from each tree.

Tasman Forestry currently defines five standard log grades:

- pruned
- Japan A
- Korea K
- domestic sawlog
- pulp

From time to time other log grades will be specified and the pre-harvest assessment data will be used to estimate volumes and characteristics of these log grades.

The pre-harvest inventory has the acronym "MARVL" standing for "Method of Assessment of Recoverable volume by Log Type". MARVL field procedure and computer analysis and software was developed by NZ Forest Research Institute in 1979. Tasman Forestry has been using it since 1981 and has a large fund of experience in its use.

Information from MARVL is a significant and critical part of Tasman Forestry's management information data base. It is used to:

- estimate the volume of harvest, by log grade, up to three years ahead of harvest
- draw up marketing plans
- set targets and prices for contract logging crews
- derive functions that allow log grade forecasts to be made for stands that are younger than those in the two to three year harvest plan
- estimate various parameters for each log grade at the time of harvest, e.g. average small end diameter, average length.

It should be noted that usually the MARVL information is "grown on" to the anticipated year of harvest before analysis is undertaken. Growing-on uses growth models developed by FRI and basically expands the diameter and height of each tree, assuming all features remain constant.

The major drawback to MARVL is the definition of a method to assess sweep, and then the ocular implementation of the chosen method. By comparison, branch class can be ocularly assessed quite accurately as can the heights to features such as forks.

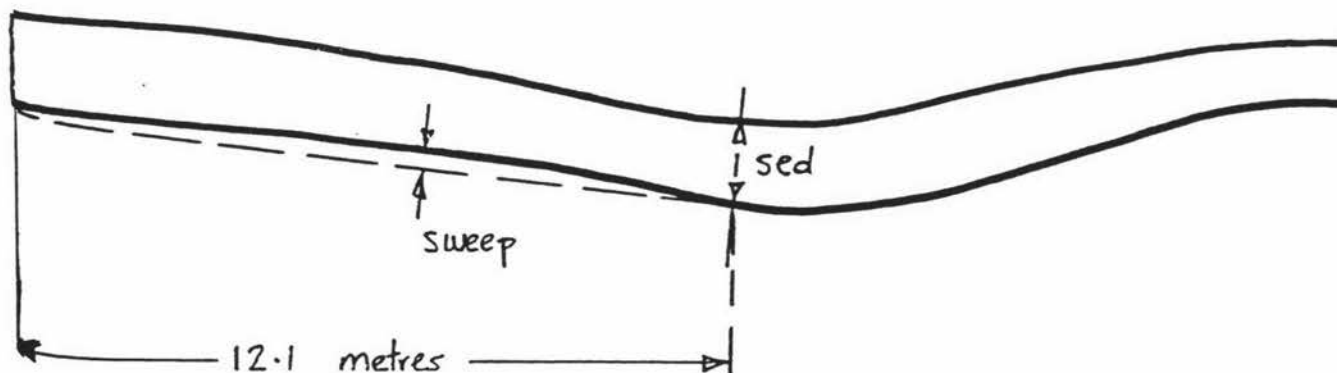
Tree Imaging

Tasman Forestry has developed MARVL toward the limit of human ability. As noted, its major weakness is the subjective assessment of sweep. In order to make a quantum improvement it is necessary to capture a 3-dimensional image of standing trees. If this can be done, the need to define a method of assessing sweep in the field disappears - the image automatically embodies all the sweep in a tree. The need to ocularly assess sweep also disappears.

The diagram below illustrates the concepts involved in analysis of the image within the computer.

View of Image in Computer (One plane only)

Log Grades	A ; B
Lengths	12.1 metres ; 8.1 metres
Minimum Small End Diameters (sed)	200mm ; 100mm
Maximum Sweep	sed/2 ; sed/1
Values	A \$100/m ³ ; B \$25/m ³



Computer tests first 12.1m section for log grade A.

- estimates sweep is it within specification?
- estimates sed is it within specification?
- if YES, goes to next section
- if NO, tries the next lower value grade (i.e. grade B) of different specification
- etc.

at Must Tree Imaging Achieve?

achievements as currently foreseen, and the working environment, are:

nding Trees

Standing trees, within normal forest stands in all their variety, are the subject. By imaging standing trees, assessment can be done one or more years ahead of harvest, there is no cutting down, and therefore likely wastage of trees, and sweep is captured without being altered by gravity acting on a felled tree.

ipling

Identical to current procedure. All trees on a 0.04 to 0.06ha plot will be imaged. Plots will sample 2% to 4% of a stand.

ameters

New Zealand plantations are now quickly moving toward a fairly uniform age of felling within the range 27-32 years. From now on mature trees will generally be around 40-45 metres in height, average 45-60 cm in diameter breast height (range 15-100), and at stockings of 200-600 stems/hectare (occasionally up to 1000 sph and reducing to a range of 200-300 sph after the turn of the century).

m Two Directions

Images should be captured from two positions at approximately 90° to one another. This will provide an outline of the shape of the stem in three dimensions.

uracy

The following limits of accuracy are desirable

- diameter ± 1 cm
- height ± 0.5 m

nches

Branches on each tree are currently classified into three size classes

0 - 7 cm	Class 1
7 - 14 cm	Class 2
> 14 cm	Class 3

For example a stem may be described as follows

5.5m	Class 1 branching
1.0 m	Class 2
12.0 m	Class 1
0.5 m	Class 3
<u>8.0 m</u>	Class 1
<u>27.0 m</u>	Total Height of Stem

It would be desirable if imaging could also estimate branch size to within ± 1 cm..

Ground Vegetation

Stands can vary widely in the density of understorey, e.g. shrubs and tree ferns up to several metres in height. As is often current practice, the understorey shrubs can conveniently be cut down before assessment takes place.

Downloading

Image data collected in the field could readily be downloaded at the end of each day to a computer based at forest HQ, as is current practice for the existing MARVL assessment. Alternatively, downloading could be done through a radio data network to HQ at any time during the day.

Number of Trees

Current pre-harvest inventory crews comprising a team of two people assess around 150-200 trees per day.

Because the capability of an imaging system is unknown at this stage, a desirable level of productivity (trees captured per person-day) can not be stated. There is a trade-off. For example, if two people can capture accurate images that provide the quantum jump in tree description, then 50 trees per two-person day may be acceptable.

At this early stage of the project, the area of prime interest is the technical ability to capture accurate images of trees.

Appendix E

System Error Calculations

This appendix provides additional information related to the discussion on the implications of image capture geometry (see section 3.7), and presents the modelling of the geometry of the mirror system to quantify the distortion introduced by a misalignment of the scanning mirror with the axis of rotation of the scanning mechanism (see section 6.3.1.2).

- Geometric sensitivity calculation
- Modelling of mirror misalignment

E.1 Tree Plane Variation

E.1.1 Errors introduced by tree displacement

Height errors caused by tree displacement for a tree displacement of $\pm 1\text{m}$, $\pm 50\text{ cm}$ and $+ 10\text{ cm}$. See section 3.7.1.1 for discussion.

Dist from tree (m)	Height of tree estimates (m)				
	0	10	20	30	40
	Height error (m) from 100 cm displacement towards scanner				
10	0.00	1.11	2.22	3.33	4.44
15	0.00	0.71	1.43	2.14	2.86
20	0.00	0.53	1.05	1.58	2.11

Dist from tree (m)	Height of tree estimates (m)				
	0	10	20	30	40
	Height error (m) from 50 cm displacement towards scanner				
10	0.00	0.53	1.05	1.58	2.11
15	0.00	0.34	0.69	1.03	1.38
20	0.00	0.26	0.51	0.77	1.03

Dist from tree (m)	Height of tree estimates (m)				
	0	10	20	30	40
	Height error (m) from 10 cm displacement towards scanner				
10	0.00	0.10	0.20	0.30	0.40
15	0.00	0.07	0.13	0.20	0.27
20	0.00	0.05	0.10	0.15	0.20

Dist from tree (m)	Height of tree estimates (m)				
	0	10	20	30	40
	Height error (m) from 50 cm displacement away from scanner				
10	0.00	-0.48	-0.95	-1.43	-1.90
15	0.00	-0.32	-0.65	-0.97	-1.29
20	0.00	-0.24	-0.49	-0.73	-0.98

Dist from tree (m)	Height of tree estimates (m)				
	0	10	20	30	40
	Height error (m) from 100 cm displacement away from scanner				
10	0.00	-0.91	-1.82	-2.73	-3.64
15	0.00	-0.63	-1.25	-1.87	-2.50
20	0.00	-0.48	-0.95	-1.43	-1.90

E.1.2 Errors introduced by measured angle variation

Height errors caused by error in measured angle under standard operating conditions for errors in angle of +3 degrees, ± 1 degree, +0.5 degree, and +0.1 degree. See section 3.7.2.1 for discussion.

Dist from tree (m)	Height of tree estimates (m)				
	0	10	20	30	40
	Height error (m) introduced by 3 degree error in O				
10	0.00	0.57	2.37	5.65	10.68
15	0.00	0.38	1.53	3.56	6.56
20	0.00	0.28	1.14	2.60	4.74

Dist from tree (m)	Height of tree estimates (m)				
	0	10	20	30	40
	Height error (m) introduced by 1 degree error in O				
10	0.00	0.18	0.73	1.66	3.01
15	0.00	0.12	0.48	1.09	1.96
20	0.00	0.09	0.36	0.81	1.45

Dist from tree (m)	Height of tree estimates (m)				
	0	10	20	30	40
	Height error (m) introduced by 0.5 degree error in O				
10	0.00	0.18	0.73	1.66	3.01
15	0.00	0.12	0.48	1.09	1.96
20	0.00	0.09	0.36	0.81	1.45

Dist from tree (m)	Height of tree estimates (m)				
	0	10	20	30	40
	Height error (m) introduced by 0.1 degree error in O				
10	0.00	0.02	0.07	0.16	0.28
15	0.00	0.01	0.05	0.11	0.19
20	0.00	0.01	0.03	0.08	0.14

Dist from tree (m)	Height of tree estimates (m)				
	0	10	20	30	40
	Height error (m) introduced by -1 degree error in O				
10	0.00	-0.17	-0.67	-1.49	-2.60
15	0.00	-0.11	-0.45	-1.01	-1.77
20	0.00	-0.09	-0.34	-0.76	-1.34

E.1.3 Errors introduced by estimated distance variation

Height errors caused by error in estimated distance from scanner to tree for standard operating conditions for an error of +0.5 m, +0.25 m, +0.1 m, and 0.02 m. See section 3.7.2.2 for discussion.

Dist from tree (m)	Height of tree estimates (m)				
	0	10	20	30	40
	Height error (m) introduced by 50 cm error in Dist.				
10	0.00	0.50	1.00	1.50	2.00
15	0.00	0.33	0.67	1.00	1.33
20	0.00	0.25	0.50	0.75	1.00

Dist from tree (m)	Height of tree estimates (m)				
	0	10	20	30	40
	Height error (m) introduced by 25 cm error in Dist.				
10	0.00	0.25	0.50	0.75	1.00
15	0.00	0.17	0.33	0.50	0.67
20	0.00	0.13	0.25	0.38	0.50

Dist from tree (m)	Height of tree estimates (m)				
	0	10	20	30	40
	Height error (m) introduced by 10 cm error in Dist.				
10	0.00	0.10	0.20	0.30	0.40
15	0.00	0.07	0.13	0.20	0.27
20	0.00	0.05	0.10	0.15	0.20

Dist from tree (m)	Height of tree estimates (m)				
	0	10	20	30	40
	Height error (m) introduced by 2 cm error in Dist.				
10	0.00	0.02	0.04	0.06	0.08
15	0.00	0.01	0.03	0.04	0.05
20	0.00	0.01	0.02	0.03	0.04

E.2 Modelling of Mirror Misalignment

If there is a misalignment between the plane of the mirror and the axis of rotation of the mechanism, a nonlinear distortion will be introduced into captured images (see figure E.1 and section 6.3.1.2). The nature of this nonlinear distortion depends on the angular position of the rotation mechanism over which the image is captured. This nonlinear distortion was measured as approximately 6.8 cm of sweep over the calibration building and was measured as significant using a laser in a laboratory situation.

This appendix presents the Matlab modelling completed to determine the nature of this distortion.

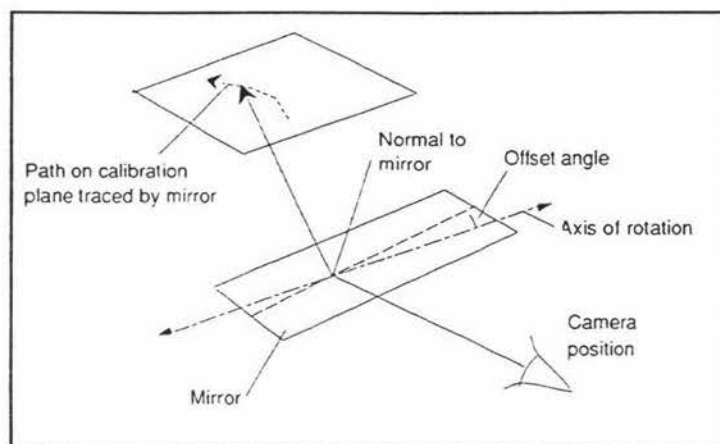


Figure E.1 - Mirror geometry

E.2.1 Analysis Approach

In a normal image capture situation, the scanner is placed at approximately 45° to the horizontal, and the image is captured over a 70° angle (see figure E.2). The bottom of the image capture may be slightly above or below the horizontal.

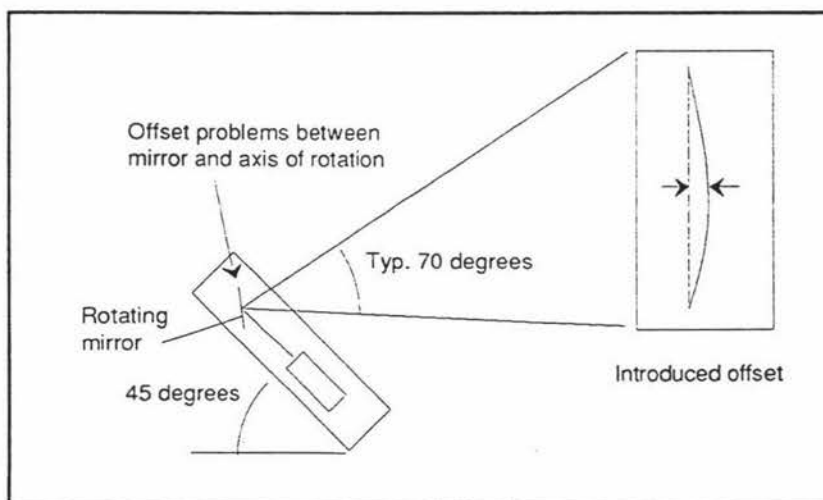


Figure E.2 - Scanner alignment during image capture

The following sequence is followed:

1. From mirror angles determine three arbitrary points on the mirror plane.
2. Using cross products of two vector differences, calculate normal to the plane.
3. Using the normal to the plane and one point calculate the equation of the plane.
4. Calculate the equation of the camera 'light ray'.
5. Calculate intercept between camera ray and mirror plane.
6. Calculate the equation of the reflected ray using the angle between the camera ray and the normal to the plane.
7. Calculate the difference in angle between reflected ray and the ideal reflected ray if there was no mirror deviation; or calculate the wall intercept in a similar fashion as steps 1 to 5.
8. Repeat steps 1-7 for a series of rotation steps representing a normal scan.

E.2.2 Mathematical Analysis

1. Three points on the mirror plane are:

Three arbitrary points on the mirror plane are defined:

$$P_1 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad P_2 = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix} \quad P_3 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad P = \begin{bmatrix} 0 & 0 & 1 \\ 1 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

To rotate the points on the mirror plane about the origin in three dimensions multiply by:

$$ofsmirr = \begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \quad \begin{array}{l} \text{Rotation about y axis} \\ \text{(mirror misalignment)} \end{array}$$

$$rotmirr = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} \quad \text{Rotation about x axis (scanning)}$$

The new loactions of P_1 to P_3 in 3D space are be found by :

$$P = rotmirr \times (ofsmirr \times P)$$

2. Find the normal to the plane

A vector normal to the plane is the cross product between any two vectors on the plane.

$$\tilde{v}_1 = P_2 - P_1, \quad \tilde{v}_2 = P_3 - P_1, \quad \tilde{N} = \tilde{v}_1 \times \tilde{v}_2.$$

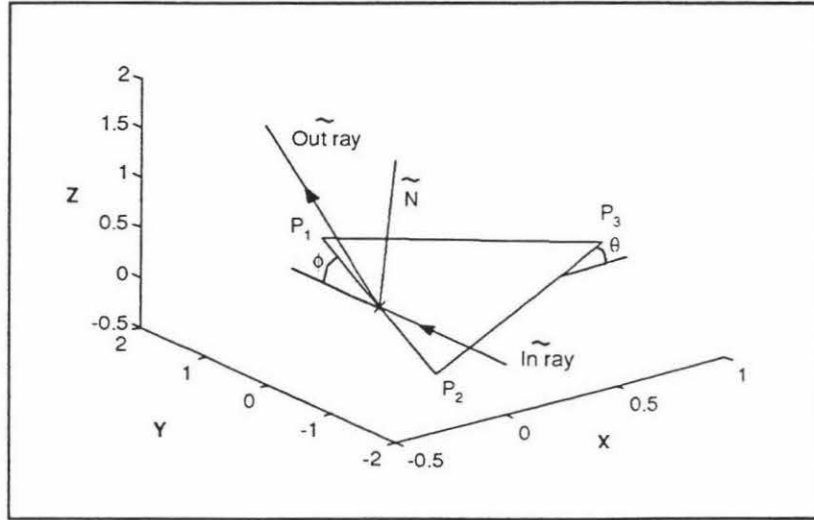


Figure E.3 - Mathematical geometry

3. Find the equation of the plane

The equation of any plane is $Ax + By + Cz + D = 0$.

$$\text{from its normal } \tilde{N} = \begin{bmatrix} A \\ B \\ C \end{bmatrix} \text{ and any point } P_0 = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$$

\Rightarrow Equation of the mirror plane : $A(x - x_0) + B(y - y_0) + C(z - z_0) = 0$

4. Find the equation of the camera ray and intercept point

The parametric equation of any line in three dimensional space is given by:

$$x = x_0 + t(x_1 - x_0), \quad y = y_0 + t(y_1 - y_0), \quad z = z_0 + t(z_1 - z_0) \text{ given any two points } P_0 \text{ and } P_1.$$

5. Find the intercept

$$\text{Using: } \begin{bmatrix} x_v \\ y_v \\ z_v \end{bmatrix} \text{ and In ray}_0 = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} \text{ to find } P_1, \text{ with } \tilde{N} = \begin{bmatrix} A \\ B \\ C \end{bmatrix} \text{ and } P_0 = \begin{bmatrix} x_p \\ y_p \\ z_p \end{bmatrix}, \text{ and}$$

substituting in the equation of the plane:

$$A(x_0 + tx_v - x_p) + B(y_0 + ty_v - y_p) + C(z_0 + tz_v - z_p) = 0$$

Rearrange to find the value of t at the intercept :

$$\therefore t(Ax_v + By_v + Cz_v) = A(x_p - x_0) + B(y_p - y_0) + C(z_p - z_0)$$

$$\therefore t = \frac{A(x_p - x_0) + B(y_p - y_0) + C(z_p - z_0)}{Ax_v + By_v + Cz_v}$$

The intercept point between the camera ray and the mirror plane can be found by substituting the value of t back into the line equations.

$$x = x_0 + t(x_1 - x_0) , y = y_0 + t(y_1 - y_0) , z = z_0 + t(z_1 - z_0)$$

6. calculate reflected ray

The equation of the reflected ray is calculated using the normal to the mirror plane and the camera ray.

If \tilde{r} = In ray, and \tilde{q} = Out ray, then

$$\tilde{r} = 2\cos(\theta)\tilde{N} - \tilde{q}$$

For any two vectors \tilde{a} and \tilde{b} with angle 2θ between them

$$a.b = |a||b|\cos(\theta) \Rightarrow \cos(\theta) = \frac{a.b}{|a||b|}$$

$$\therefore \text{Vector representing reflected ray} = \tilde{r} = 2 \left(\frac{\tilde{q} \cdot \tilde{N}}{|\tilde{q}||\tilde{N}|} \right) \tilde{N} - \tilde{q}$$

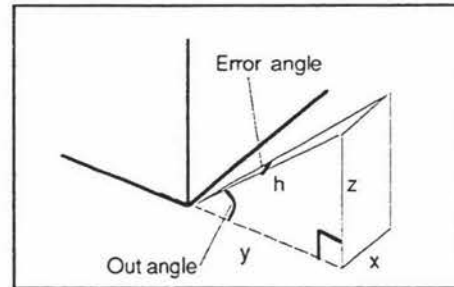
7. Calculate error angle

Calculate the difference in spatial angle between the reflected ray and theoretical reflected ray if there was no mirror deviation. In this case:

$$\text{out angle} = \text{atan}\left(\frac{z}{y}\right)$$

$$h = \frac{y}{\cos(\text{out angle})}$$

$$\text{error angle} = \tan^{-1}\left(\frac{x}{h}\right)$$



To calculate the path traced by the reflected ray on the calibration plane the same sequence of steps (steps 1 to 5) can be used to find the intercept point between the reflected ray and the calibration plane (see program listing in section E.2.4).

E.2.3 Results

The magnitude and shape of the deviation introduced varies with the position of rotation of the mirror. If the plane of the mirror is parallel to the camera ray the ray is not reflected, and thus not deviated. If the plane of the mirror is perpendicular the camera ray the deviation is at a maximum of twice the offset angle.

Two scenareos were modelled. The first a typical image capture situation where the object (at the calibration plane) being imaged was 15 m away from the scanner, the second was the controlled lab situation under which the deviation was measured.

The results showed that a misalignment of 0.8° (1 mm at one end of the mirror) introduced an apparent sweep with maximum deflection of 7 cm (see figure E.4). To reduce this to an acceptable 1 cm the mirror must be aligned to 0.1° (0.14 mm at one end of the mirror).

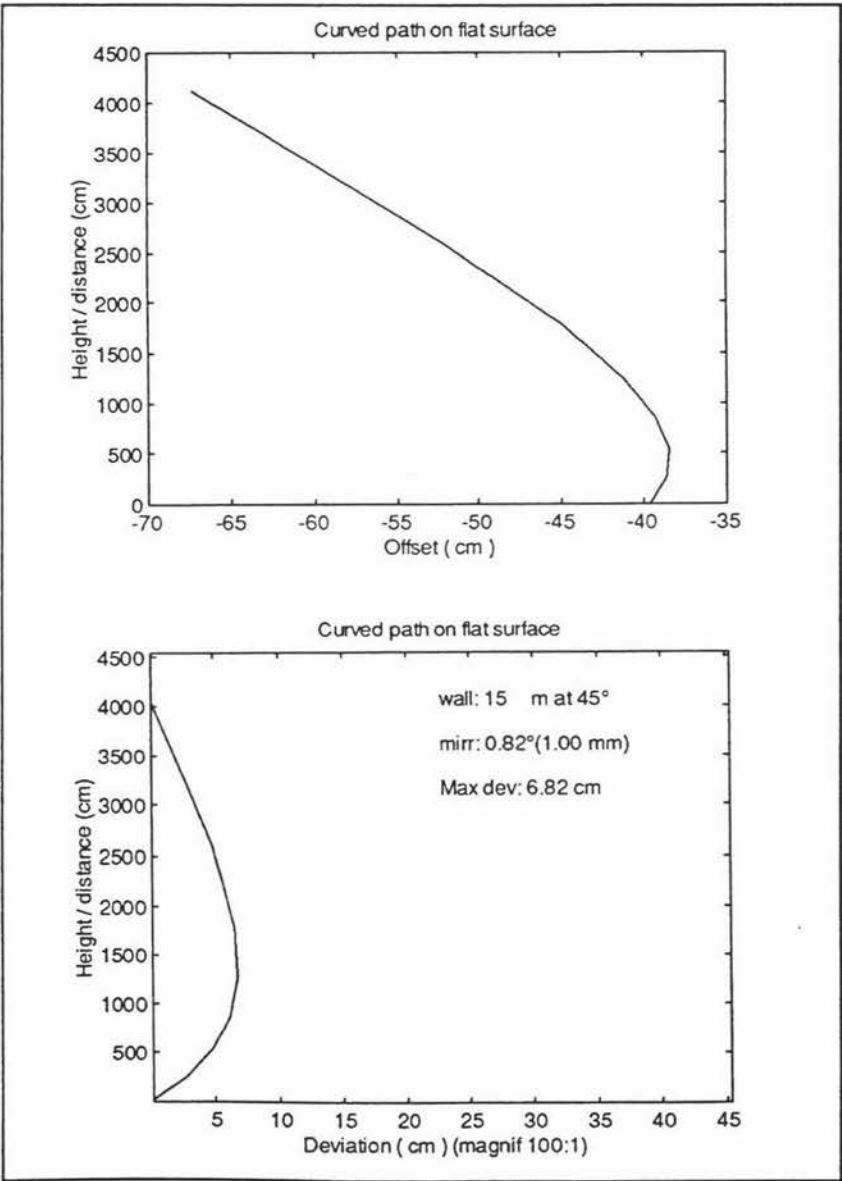


Figure E.4 - Typical image capture with 0.82° mirror deviation

E.2.4 Matlab Model

```

%=====
% Marijn Weehuizen 12 September 1995
%=====
% Model of the mirror mechanism to
% investigate a curved deviation in the
% captured images.
%=====
% Output:
% Store( loop errangle outangle wally wallx)
%=====

clear

%=====
% Define constants
%=====
% Angle of wall ( 0 = lab stsetup, 45 = vet tower setup)
% Distance to wall (cm)

% wallangle =45; walldist = 1500;      % Vet tower
% wallangle =0; walldist = 210;        % Lab setup
% wallangle =0; walldist = 1500;      % Tests

% ofsangle = atan(0.1/7)*180/pi;      % Mirror ofset angle
% ofsangle = 0;                      % Mirror ofset angle

loopstop = 90-wallangle/2; loopangle = 35;
step = 5;                            % Rotation of mirror steps
x=1; y=2; z=3;
count=1;
ofsangle = ofsangle / 180 * pi;
wallangle= wallangle / 180*pi * (-1) ;

%=====
% Set up wall
%=====

d1 = cos(-wallangle)*walldist;
d2 = cos(90/180*pi - (-wallangle))*(tan(-wallangle)*walldist);
wdist = d1+d2;

%=====
for loop=loopstop-loopangle:step:loopstop
%for loop=67.5:step:67.5
rotangle = loop / 180 * pi;

%=====
% Setup light ray
%=====
hold off

%rofs=tan(0/180*pi)*2;
%ray=[rofs; 2; 0];
%ray0=[-rofs; -2; 0];

rofs=tan(1/180*pi)*2;
ray=[rofs; 2; 0];
ray0=[0; -2; 0];

```

```

plot3([ray0(1) ray0(1)+ray(1)], [ray0(2) ray0(2)+ray(2)], [ray0(3) ray0(3)+ray(3)])
hold on

plot3(1,1,1);

%-----
%Set up mirror plane
%-----

p=[0 0 1;
   1 -1 0;
   0 0 0];
po = 3.5;
pofs=[ po po po;
        0 0 0;
        0 0 0];

ofsmirr = [      cos(ofsangle)    0      (-sin(ofsangle));
             0                    1      0;
             sin(ofsangle)    0      cos(ofsangle)];

rotmirr = [      1                0      0;
             0                cos(rotangle)  (-sin(rotangle));
             0                sin(rotangle)  cos(rotangle)];

p=ofsmirr*p;
p=rotmirr*p;
p=p-pofs;
plot3([p(1,:) p(1,1) ], [p(2,:) p(2,1)], [p(3,:) p(3,1) ] )
%-----
% find normal
%-----
v1 = p(:,2)-p(:,1);
v2 = p(:,3)-p(:,1);
N=cross(v1,v2);

%-----
% find equation of plane
%-----
%t=( A(xp - x0) + B(yp - y0) + C(zp - z0) )/(A*xv + A*xv + A*xv)

t=( N(1)*(p(1,1) - ray0(1)) + N(2)*(p(2,1) - ray0(2)) + N(3)*(p(3,1) - ray0(3)) )/(N(1)*ray(1) +
    N(2)*ray(2) + N(3)*ray(3));

intpnt=[ray0(1) + t * ray(1);
        ray0(2) + t * ray(2);
        ray0(3) + t * ray(3)];

plot3(intpnt(1), intpnt(2), intpnt(3), 'cx');

plot3([intpnt(1) intpnt(1)+N(1)], [intpnt(2) intpnt(2)+N(2)], [intpnt(3) intpnt(3)+N(3)], 'c')

%pause
%-----
% Calculate reflected ray of light
%-----

outray = -(2*(dot(ray,N)/(sqrt(sum(ray.^2))*sqrt(sum(N.^2)))) * N - ray);

plot3([intpnt(x) intpnt(x)+outray(x)], [intpnt(y) intpnt(y)+outray(y)], [intpnt(z) intpnt(z)+outray(z)])

%-----

```

```

% Calculate offset angle
%-----
mirrangle = loop

outangle = atan(outray(z)/outray(y))*180/pi;
if outangle<0 outangle = 180 + outangle; end
outangle = 180-outangle;

h = outray(2)/cos(outangle/180*pi);
errangle = atan(outray(x)/h)*180/pi;

store(count,1:3)=[loop errangle outangle];

%=====
% Set up wall
%-----

w=[      0          1          0;
    -wdist      -wdist      -wdist;
      0          0          2];
w0=[0; -wdist; 0];

rotwall = [      1          0          0;
            0      cos(wallangle)  (-sin(wallangle));
            0      sin(wallangle)  cos(wallangle)];

w(:,1)=w(:,1)-w0; w(:,2)=w(:,2)-w0; w(:,3)=w(:,3)-w0;
w=rotwall*w;
w(:,1)=w(:,1)+w0; w(:,2)=w(:,2)+w0; w(:,3)=w(:,3)+w0;

plot3([w(1,:) w(1,1) ],[w(2,:) w(2,1)],[w(3,:) w(3,1) ] )
%-----
% find normal
%-----
wv1 = w(:,2)-w(:,1);
wv2 = w(:,3)-w(:,1);
wN=cross(wv1,wv2);

%-----
% find equation of plane
%-----
%t=( A(xp - x0) + B(yp - y0) + C(zp - z0) )/(A*xv + A*yv + A*zv)

t=( wN(x)*(w(x,1) - intpnt(x)) + wN(y)*(w(y,1) - intpnt(y)) + wN(z)*(w(z,1) - intpnt(z)) )
/(wN(x)*outray(x) + wN(y)*outray(y) + wN(z)*outray(z));

wallpnt=[intpnt(x) + t * outray(x);
        intpnt(y) + t * outray(y);
        intpnt(z) + t * outray(z)];

plot3(wallpnt(x), wallpnt(y), wallpnt(z),'cx');

plot3([wallpnt(x) wallpnt(x)+wN(x)], [wallpnt(y) wallpnt(y)+wN(y)], [wallpnt(z) wallpnt(z)+wN(z)], 'c')

%-----

wp=wallpnt-w0;
wp=rotwall'*wp;
wallx=wp(x); wally=wp(z)-walldist*tan(-wallangle);
store(count,4:5)=[wally wallx];

%-----

```

```

%pause
count=count+1;
end

%-----
% Calculate deviation
%-----

% not required

%-----
% Correct curve
%-----

loop = 1;
maxloop = size(store,1)

corrangle = atan ((store(1,5) - store(maxloop,5)) / (store(1,4) - store(maxloop,4)) );
corrangle*180/pi
rotcorr = [      cos(corrangle)   (-sin(corrangle));
            sin(corrangle)   cos(corrangle)];

for loop=1:maxloop
temp(x,1)=store(loop,5)-store(maxloop,5); temp(y,1)=store(loop,4)-store(maxloop,4);
temp=rotcorr*temp;
%temp(x)=temp(x)+store(maxloop,5);
temp(y)=temp(y)+store(maxloop,4);

store(loop,6:7)=[temp(y,1) temp(x,1)];
end

maxdev = sign(store(5,7))*max(abs(store(:,7)))

%=====
% Plot graphs
%-----

hold off

%plot(store(:,3),store(:,2))
%title('Deviation of ray from view plane for mirror deviation of 5deg')
%xlabel('Angle between mirror incident light and reflected light (degrees)')
%ylabel('degrees')

plot(store(:,5),store(:,4),'w')
title('Curved path on flat surface')
xlabel('Offset ( cm )')
ylabel('Height / distance (cm)')

pause

plot(store(:,7),store(:,6),'w')
title('Curved path on flat surface')
xlabel('Deviation ( cm ) (magnif 100:1)')
ylabel('Height / distance (cm)')

ymin=store(maxloop,6);
ymax= max(store(:,6))*1.1;

if maxdev>0
% AXIS([ymin ymax/100 ymin ymax]);
AXIS([0 30 ymin ymax]);
text(0.5*ymax/100, 0.9*ymax, sprintf('wall: %dm at %.0f∞',walldist, wallangle*180/pi*(-1)));
text(0.5*ymax/100, 0.8*ymax,sprintf('mirr: %.2f∞(%.2f mm)',ofsangle*180/pi,tan(ofsangle)*70));

```



```
text(0.5*yymax/100, 0.7*yymax, sprintf('Max dev: %.2f cm', maxdev));
else
  AXIS([-yymax/100 -ymin ymin ymax]);
  text(-0.9*yymax/100, 0.9*yymax, sprintf('wall: %dm at %.0f°', walldist, wallangle*180/pi*(-1)));
  text(-0.9*yymax/100, 0.8*yymax, sprintf('mirr: %.2f°(%.2f mm)', ofsangle*180/pi, tan(ofsangle)*70));
  text(-0.9*yymax/100, 0.7*yymax, sprintf('Max dev: %.2f cm', maxdev));
end

store
```

Appendix F

TreeScan Component List

F.1 TreeScan System Component List

The TreeScan system has been designed as a complete working system and consists of the following components:

1. Portable computer (Macintosh PowerBook 520c, 160 MByte hard disk, 20 MByte RAM)
2. TreeScan scanner
3. Calibration rod
4. Tripod
5. SCSI cable (computer to scanner)
6. Power cable (batteries to scanner)
7. Set of two batteries (for scanner)
8. Set of two carrying cases
9. Scope sight for scanner
10. Digital level to measure alignment

The above items are required during image capture. In addition to this, system comes complete with:

- Charger for scanner batteries
- Charger for the computer's internal batteries
- Documentation

A. TreeScan Operator Manual

B. TreeScan Technical Reference Manual

The computer and batteries are contained in the computer carry case. All other components from 1 to 10 are carried in the scanner carrying case.

F.2 TreeScan Scanner Component List

The TreeScan scanner has been custom built and contains the following components:

- Chassey (made from 45 cm of 150 x 75 mm Al channel)
- Rotating mirror mechanism
 - Brass shaft
 - Brass wormwheel with steel rotation shaft
 - 2 x pivots mounted on ball bearing
 - Stepper motor (3.75 degree / step)
 - 2 x opto interrupter position sensors
- Line scan camera (Loral Fairchild CAM 1301R)
- Lens (Fixed 75 mm focal length, manual aperture, motorised focus)
- Focus Mechanism
 - Stepper motor (3.75 degree / step)
 - Teflon wormwheel drive mechanism
- Controller board

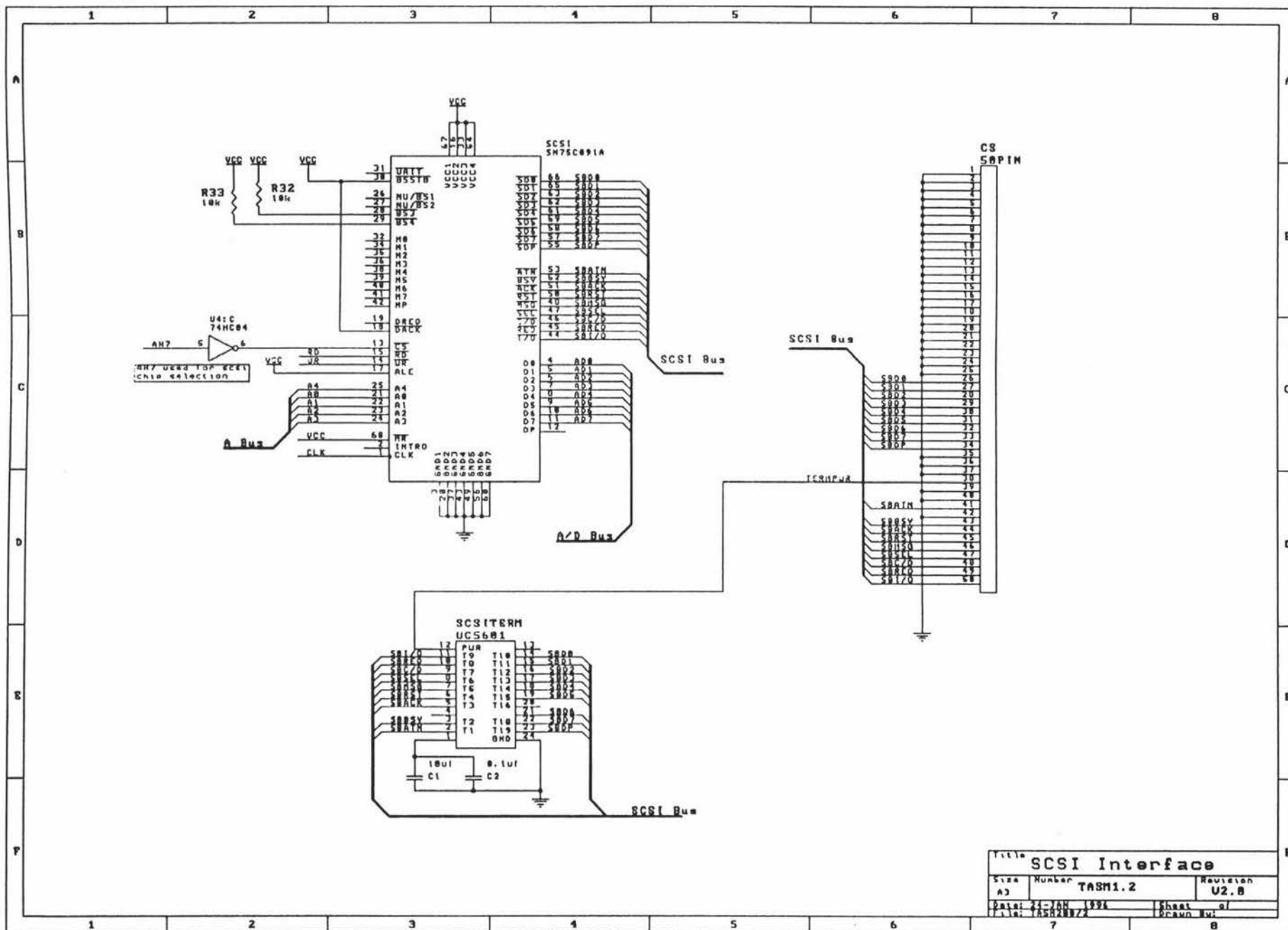
Siemens 80C517A microcontroller	74HC573 address latch
27256 EPROM (32 kByte, 170 nS)	62256 RAM (32 kByte)
16 MHz oscillator	
SN75C091A SCSI bus controller	UC5601 SCSI bus terminator
2 x L297 stepper motor controller	2 x 74HC04 inverter
2 x L298N stepper motor driver	74HC08 And gate
PowerBox 12V to 15V DC-DC converter	
LM2938-5 low dropout 5V regulator	LM 317L +15V regulator
Relay	LM 337L -15V regulator
Various resistors	Various capacitors
Decoupling capacitors	

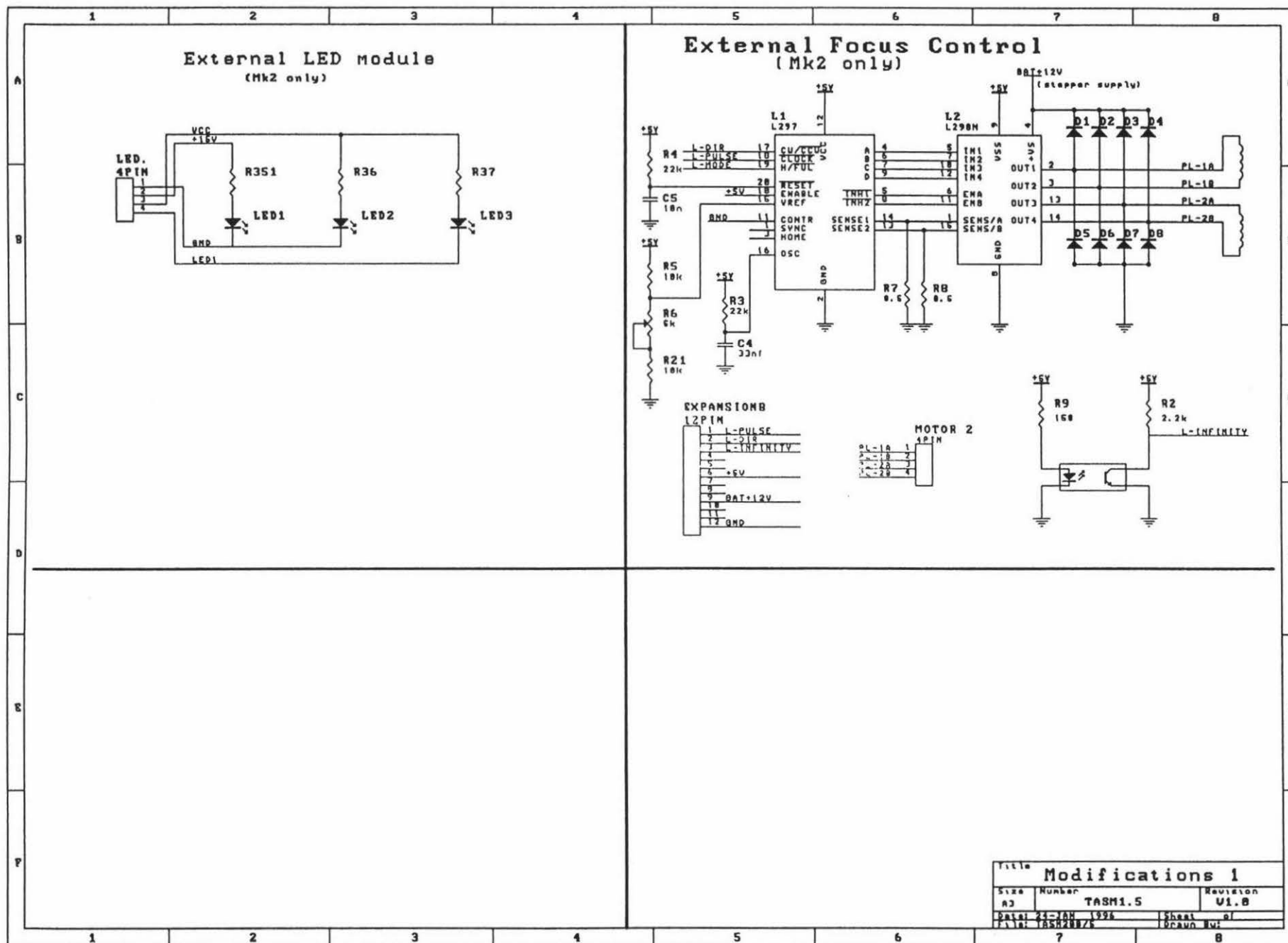
Appendix G

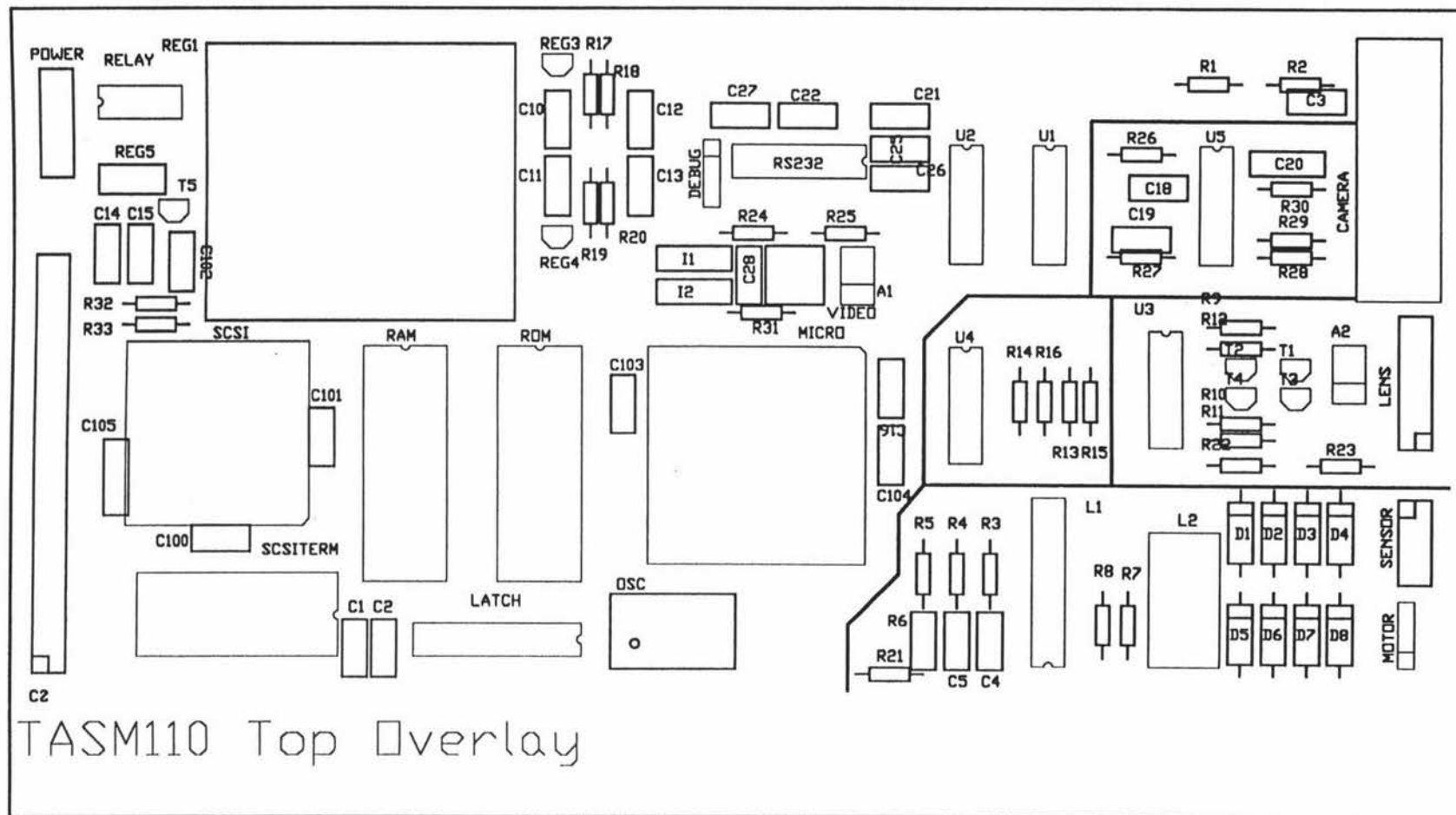
TreeScan Schematics and Controller Board Layout

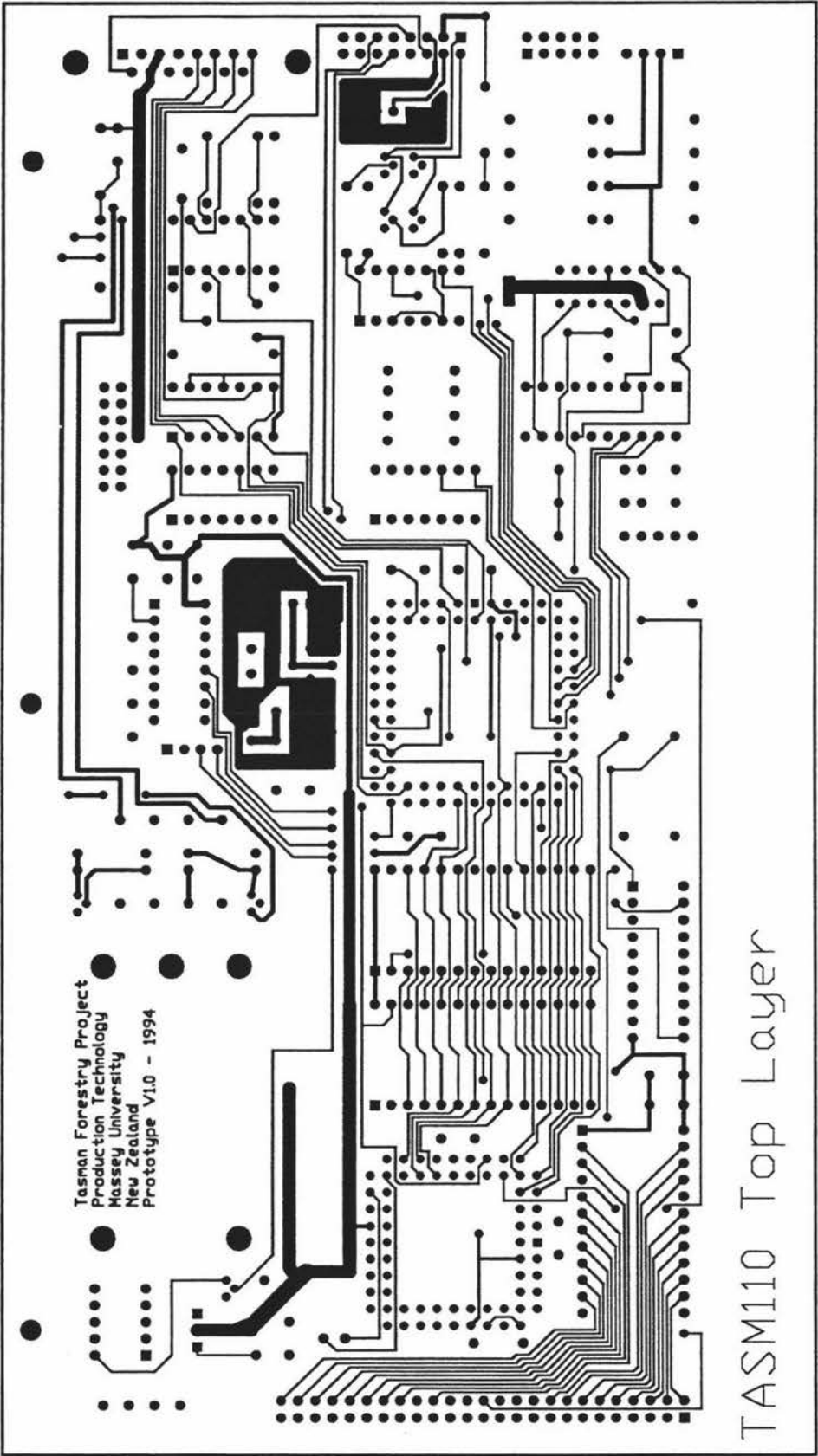
This appendix provides full TreeScan schematics and controller board PCB layout.

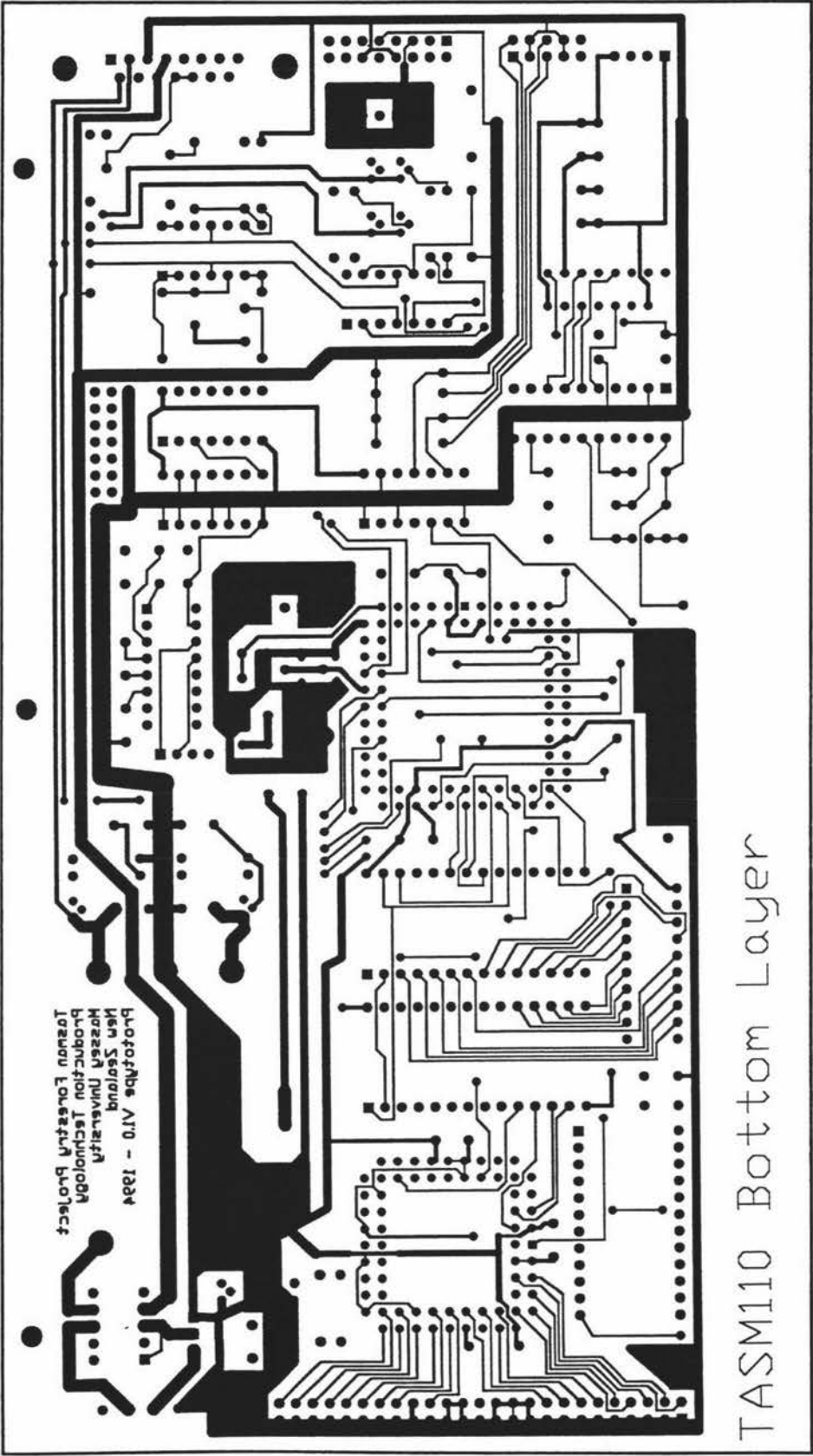
- The schematics are of the controller board in the Mk2 TreeScan system.
- To develop the Mk2 controller board, modifications were wired on to the back of the PCB used for the Mk1 system. Hence the controller board PCB layouts provided are as the Mk1 system was manufactured.











Appendix H

Microcontroller Specifications and Memory Space Organisation

H.1 80C517A Microcontroller Features

**High-Performance
8-Bit CMOS Single-Chip Microcontroller**

SAB 80C517A / 83C517A-5

Preliminary

**SAB 83C517A-5
SAB 80C517A**

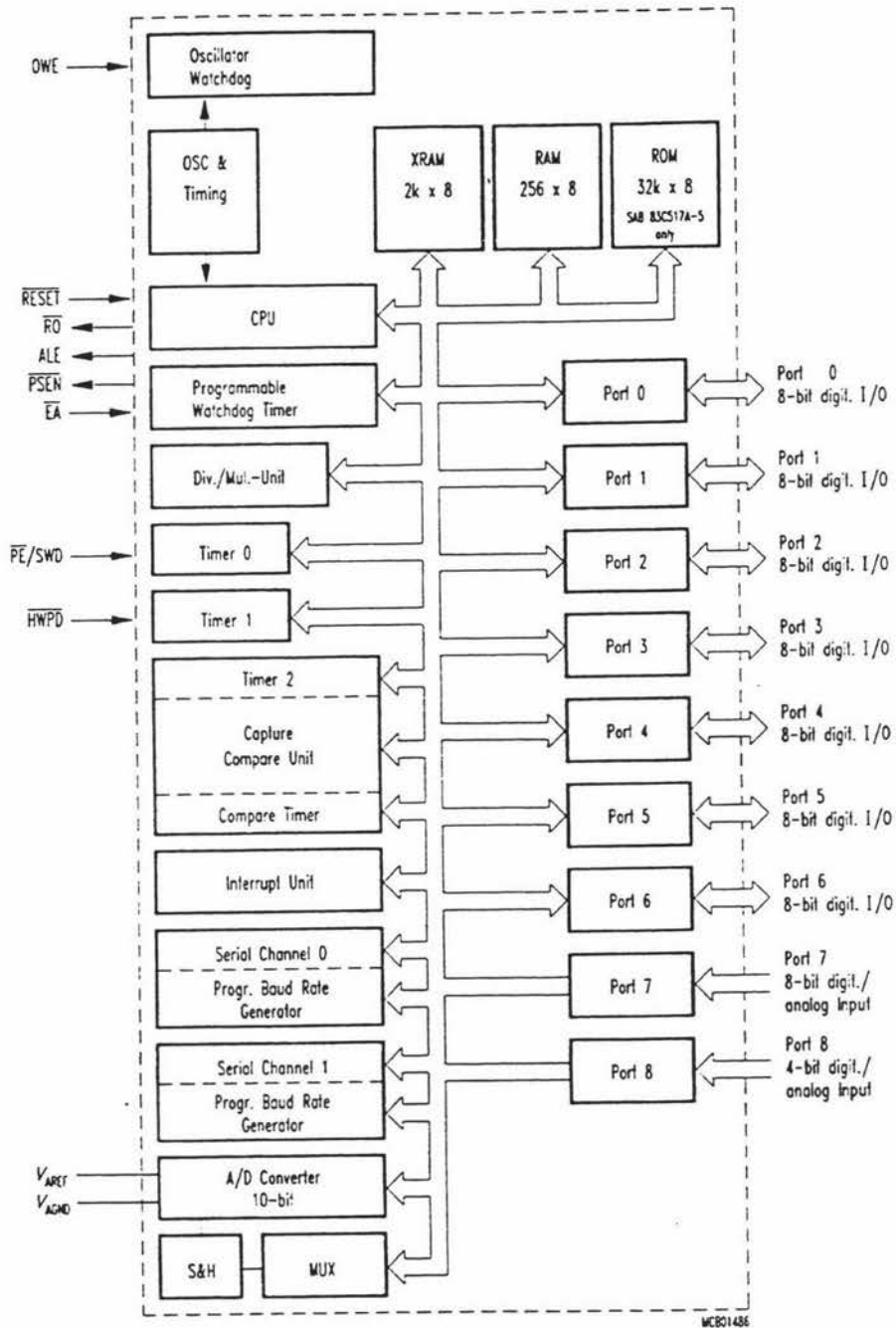
Microcontroller with factory mask-programmable ROM
Micro controller for external ROM

- SAB 80C517A / 83C517A-5, up to 18 MHz operation
- 32 K × 8 ROM (SAB 83C517A-5 only, ROM-Protection available)
- 256 × 8 on-chip RAM
- 2 K × 8 on-chip RAM (XRAM)
- Superset of SAB 80C51 architecture:
 - 1 µs instruction cycle time at 12 MHz
 - 666 ns instruction cycle time at 18 MHz
 - 256 directly addressable bits
 - Boolean processor
 - 64 Kbyte external data and program memory addressing
- Four 16-bit timer/counters
- Powerful 16-bit compare/capture unit (CCU) with up to 21 high-speed or PWM output channels and 5 capture inputs
- Versatile "fail-safe" provisions
- Fast 32-bit division, 16-bit multiplication, 32-bit normalize and shift by peripheral MUL/DIV unit (MDU)
- Eight data pointers for external memory addressing
- Seventeen interrupt vectors, four priority levels selectable
- Genuine 10-bit A/D converter with 12 multiplexed inputs
- Two full duplex serial interfaces with programmable Baudrate-Generators
- Fully upward compatible with SAB 80C515, SAB 80C517, SAB 80C515A
- Extended power saving mode
- Fast Power-On Reset
- Nine ports: 56 I/O lines, 12 input lines
- Three temperature ranges available:
 - 0 to 70 °C (T1)
 - 40 to 85°C (T3)
 - 40 to 110°C (T4)
- Plastic packages: P-LCC-84, P-MRFP-100

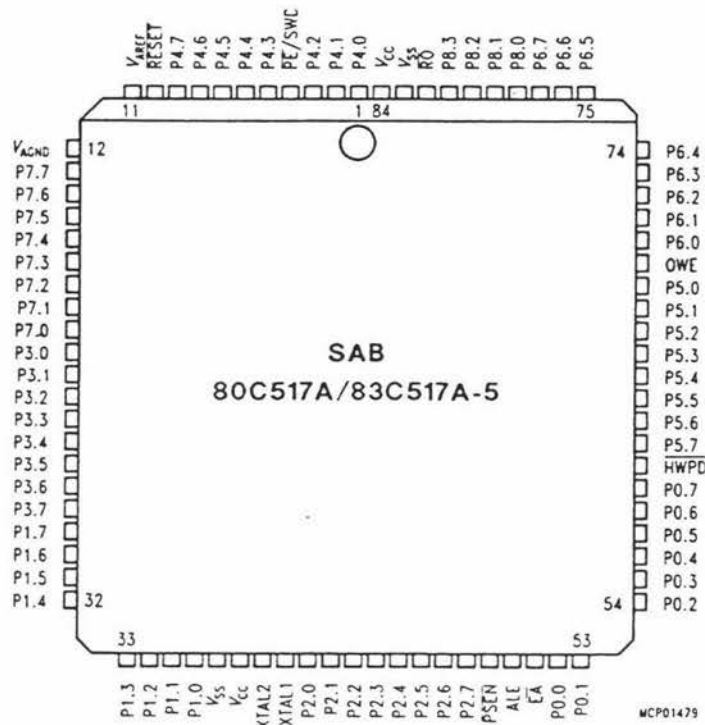
The SAB 80C517A/83C517A-5 is a high-end member of the Siemens SAB 8051 family of microcontrollers. It is designed in Siemens ACMOS technology and based on SAB 8051 architecture. ACMOS is a technology which combines high-speed and density characteristics with low-power consumption or dissipation. package (P-LCC-84) and in a 100-pin plastic quad flat package (P-MRFP-100).

While maintaining all the SAB 80C517 features and operating characteristics the SAB 80C517A is expanded in its "fail-safe" characteristics and timer capabilities. The SAB 80C517A is identical with the SAB 83C517A-5 except that it lacks the on-chip program memory. The SAB 80C517A / 83C517A-5 is supplied in a 84-pin plastic leaded chip carrier package (P-LCC-84) and in a 100-pin plastic quad flat package (P-MRFP-100).

H.2 Microcontroller Block Diagram



H.3 Microcontroller Pin Configuration



H.3 Microcontroller Electrical Characteristics

Absolute Maximum Ratings

Ambient temperature under bias	- 40 to 110° C
Storage temperature	- 65 to 150 °C
Voltage on V _{CC} pins with respect to ground (V _{SS})	- 0.5 V to 6.5 V
Voltage on any pin with respect to ground (V _{SS})	- 0.5 to V _{CC} +0.5 V
Input current on any pin during overload condition	- 10mA to +10mA
Absolute sum of all input currents during overload condition	100mA
Power dissipation	1 W

Note Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage of the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for longer periods may affect device reliability. During overload conditions (V_{IN} > V_{CC} or V_{IN} < V_{SS}) the Voltage on V_{CC} pins with respect to ground (V_{SS}) must not exceed the values defined by the absolute maximum ratings.

DC Characteristics
 $V_{CC} = 5\text{ V} \pm 10\%, -15\%; V_{SS} = 0\text{ V}$
 $T_A = 0\text{ to }70\text{ }^\circ\text{C}$ for the SAB 80C517A/83C517A-5

 $T_A = -40\text{ to }85\text{ }^\circ\text{C}$ for the SAB 80C517A-T3/83C517A-5-T3

 $T_A = -40\text{ to }110\text{ }^\circ\text{C}$ for the SAB 80C517A-T4/83C517A-5-T4

Parameter	Symbol	Limit Values		Unit	Test condition
		min.	max.		
Input low voltage (except EA, RESET, HWRPD)	V_{IL}	-0.5	$0.2 V_{CC} - 0.1$	V	—
Input low voltage ($\overline{\text{EA}}$)	V_{IL1}	-0.5	$0.2 V_{CC} - 0.3$	V	—
Input low voltage (HWRPD, RESET)	V_{IL2}	-0.5	$0.2 V_{CC} + 0.1$	V	—
Input high voltage (except RESET, XTAL2 and HWRPD)	V_{IH}	$0.2 V_{CC} + 0.9$	$V_{CC} + 0.5$	V	—
Input high voltage to XTAL2	V_{IH1}	$0.7 V_{CC}$	$V_{CC} + 0.5$	V	—
Input high voltage to RESET and HWRPD	V_{IH2}	$0.6 V_{CC}$	$V_{CC} + 0.5$	V	—

DC Characteristics (cont'd)

Parameter	Symbol	Limit Values		Unit	Test condition
		min.	max.		
Output low voltage (ports 1, 2, 3, 4, 5, 6)	V_{OL}	—	0.45	V	$I_{OL} = 1.6\text{ mA}^{(1)}$
Output low voltage (ports ALE, PSEN, RO)	V_{OL1}	—	0.45	V	$I_{OL} = 3.2\text{ mA}^{(1)}$
Output high voltage (ports 1, 2, 3, 4, 5, 6)	V_{OH}	2.4 $0.9 V_{CC}$	— —	V V	$I_{OL} = -80\text{ }\mu\text{A}$ $I_{OL} = -10\text{ }\mu\text{A}$
Output high voltage (port 0 in external bus mode, ALE, PSEN, RO)	V_{OH1}	2.4 $0.9 V_{CC}$	— —	V V	$I_{OL} = -800\text{ }\mu\text{A}^{(2)}$ $I_{OL} = -80\text{ }\mu\text{A}^{(2)}$
Logic input low current (ports 1, 2, 3, 4, 5, 6)	I_{IL}	-10	-70	μA	$V_{IN} = 0.45\text{ V}$
Logical 1-to-0 transition current (ports 1, 2, 3, 4, 5, 6)	I_{TL}	-65	-650	μA	$V_{IN} = 2\text{ V}$
Input leakage current (port 0, EA, ports 7, 8, HWRPD)	I_{LI}	—	± 100	nA	$0.45 < V_{IN} < V_{CC}$
			± 150	nA	$0.45 < V_{IN} < V_{CC}$ $T_A > 100\text{ }^\circ\text{C}$
Input low current to RESET for reset	I_{IL2}	-10	-100	μA	$V_{IN} = 0.45\text{ V}$
Input low current (XTAL2)	I_{IL3}	—	-15	μA	$V_{IN} = 0.45\text{ V}$
Input low current ($\overline{\text{PE}}$ /SWD, OWE)	I_{IL4}	—	-20	μA	$V_{IN} = 0.45\text{ V}$
Pin capacitance	C_{IO}	—	10	pF	$f_C = 1\text{ MHz}$ $T_A = 25\text{ }^\circ\text{C}$
Power supply current:					
Active mode, 12 MHz ⁽⁷⁾	I_{CC}	—	28	mA	$V_{CC} = 5\text{ V}^{(4)}$
Active mode, 18 MHz ⁽⁷⁾	I_{CC}	—	37	mA	$V_{CC} = 5\text{ V}^{(4)}$
Idle mode, 12 MHz ⁽⁷⁾	I_{CC}	—	24	mA	$V_{CC} = 5\text{ V}^{(5)}$
Idle mode, 18 MHz ⁽⁷⁾	I_{CC}	—	31	mA	$V_{CC} = 5\text{ V}^{(5)}$
Slow down mode, 12 MHz	I_{CC}	—	12	mA	$V_{CC} = 5\text{ V}^{(6)}$
Slow down mode, 18 MHz	I_{CC}	—	16	mA	$V_{CC} = 5\text{ V}^{(6)}$
Power Down Mode	I_{PD}	—	50	μA	$V_{CC} = 2\text{...}5.5\text{ V}^{(3)}$

A/D Converter Characteristics

$V_{CC} = 5\text{ V} + 10\%, -15\%; V_{SS} = 0\text{ V}$
 $V_{AREF} = V_{CC} \pm 5\%; V_{AGND} = V_{SS} \pm 0.2\text{ V};$
 $T_A = 0\text{ to }70\text{ }^\circ\text{C}$ for the SAB 80C517A/83C517A-5
 $T_A = -40\text{ to }85\text{ }^\circ\text{C}$ for the SAB 80C517A-T3/83C517A-5-T3
 $T_A = -40\text{ to }110\text{ }^\circ\text{C}$ for the SAB 80C517A-T4/83C517A-5-T4

Parameter	Symbol	Limit values			Unit	Test condition
		min.	typ.	max.		
Analog input capacitance	C_I		25	70	pF	
Sample time (inc. load time)	T_S			$4\ t_{CY}^{1)}$	μS	²⁾
Conversion time (inc. sample time)	T_C			$14\ t_{CY}^{1)}$	μS	³⁾
Total unadjusted error	TUE			± 2	LSB	$V_{AREF} = V_{CC}$ $V_{AGND} = V_{SS}$
V_{AREF} supply current	I_{REF}		± 20		μA	⁴⁾

¹⁾ $t_{CY} = (8 \cdot 2^{ADCL}) / f_{OSC}$; ($t_{CY} = 1 / f_{ADC}$; $f_{ADC} = f_{OSC} / (8 \cdot 2^{ADCL})$)
²⁾ This parameter specifies the time during the input capacitance C_I can be charged/discharged by the external source. It must be guaranteed, that the input capacitance C_I is fully loaded within this time. $4TCY$ is $2\ \mu\text{s}$ at the $f_{OSC} = 16\text{ MHz}$. After the end of the sample time T_S changes of the analog input voltage have no effect on the conversion result.
³⁾ This parameter includes the sample time T_S . $14TCY$ is $7\ \mu\text{s}$ at $f_{OSC} = 16\text{ MHz}$.
⁴⁾ The differential impedance r_D of the analog reference source must be less than $1\text{ K}\Omega$ at reference supply voltage.

H.4 Memory space organisation

The memory space organisation of the 80C517 CPU is complicated. The 80C517 CPU has separate address spaces for program and data memory, and manipulates operands in the following four address spaces:

- Up to 64 kByte of program memory
- Up to 64 kByte of external data memory
- 256 bytes of internal data memory
- 128 byte special function register area

Program memory can be external (EPROM) or up to 32 kByte on the micro controller chip determined by the state of the EA pin during powerup. The 80C517A also has 2 kByte on chip XRAM. The XRAM is accessed using identical instructions to accessing external RAM but with bit 1 of SYSCON register set.

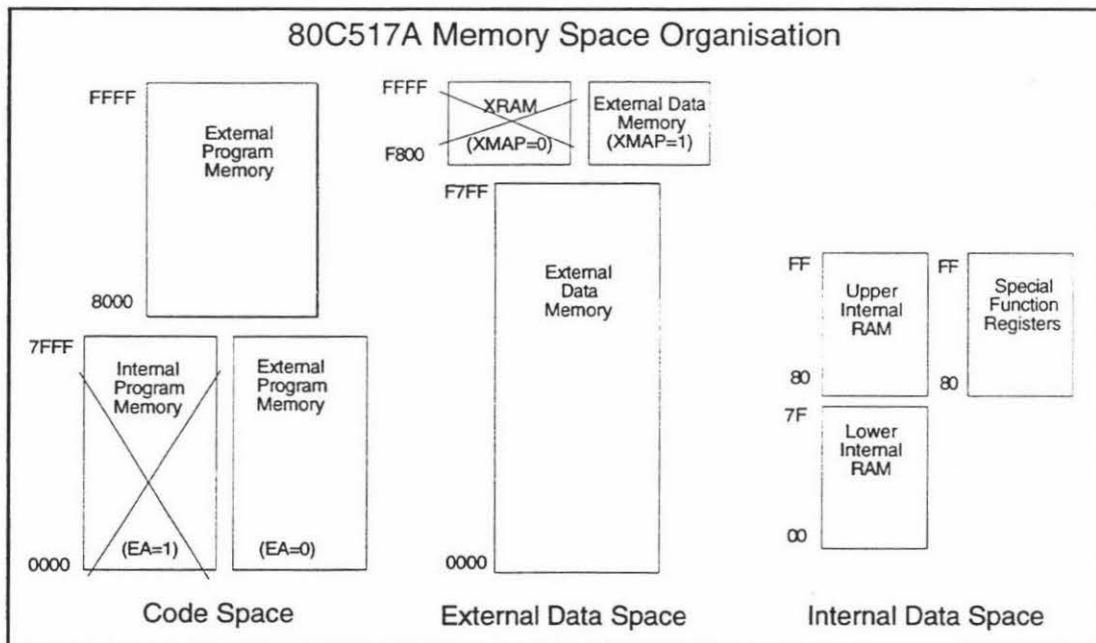


Figure H.1 - Microcontroller memory space organisation.

All registers, (except the program counter and four general purpose register banks), reside in the special function register (SFR) area. The SFR's include arithmetic registers, pointers and registers to provide an interface between on chip peripherals (eg. IO Ports). Registers which lie on 8 Byte boundaries are bit addressable.

There is an address overlap between the upper 128 bytes of internal RAM and the SFR's. The addressing mode used determines whether the SFR's are addressed or

whether internal RAM is addressed. The internal RAM contains four banks of registers and 128 bit addressable bits overlapping internal RAM. The stack pointer is initialised to 08h in internal RAM after reset.

The TreeScan scanner microcontroller uses the following sections of the 80C517A memory space (see figure 4.7) :

- 32K byte EPROM to store the microcontroller code and A/D lookup tables
- 32K byte RAM of which 1K byte is used to buffer the SCSI transfer
- SCSI controller registers repeatedly mapped into the top 32K bytes of external data memory
- the lower internal RAM (for working variables)
- the special function registers

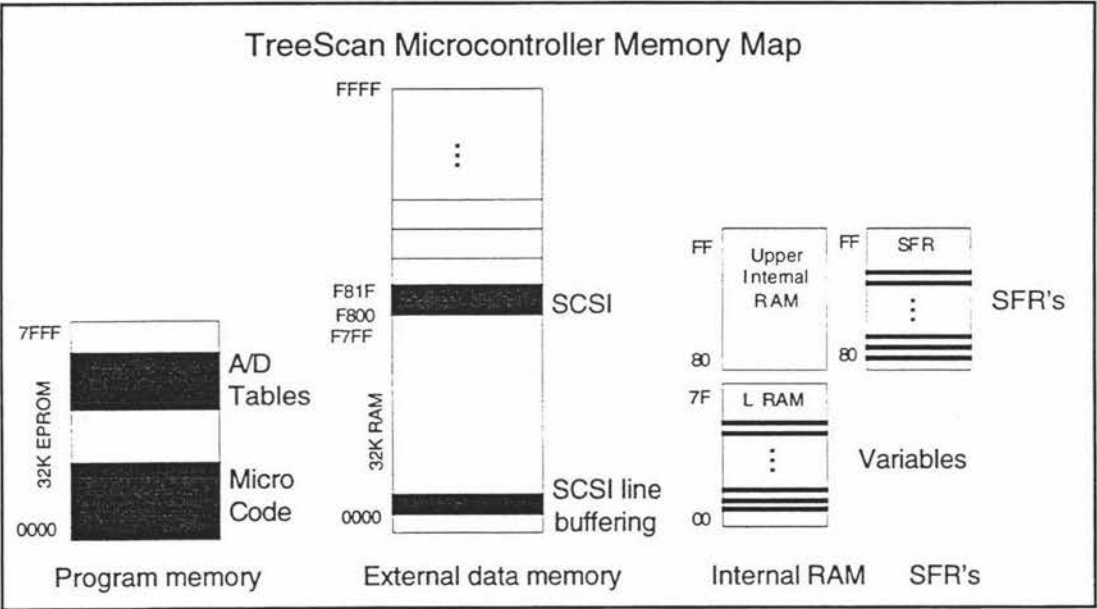


Figure H.2 - TreeScan Microcontroller memory map

Appendix I

Additional SCSI Interface Specifications

Appendix I provides SCSI interface specifications additional to the discussion in sections 4.4.1 and 5.2.4. Note that this information relates to the implementation of a SCSI 1 system.

I.1 SCSI Bus Phases

The SCSI contains eight distinct phases of the SCSI bus. The SCSI bus can only be in one of these phases at any one time. Each of the eight possible phases has a specific purpose:

- **BUS FREE phase**
The BUS FREE phase is used to indicate that no SCSI device is actively using the SCSI bus and that it is available to subsequent to devices.
- **Arbitration phase**
The ARBITRATION phase allows one device to gain control of the SCSI bus so that it can assume the role of an initiator or target. The device with the highest ID number wins the arbitration.
- **Selection phase**
The SELECTION phase allows an initiator to select a target for the purpose of initiating some target function. eg. a data transfer command.
- **Reselection phase**
In systems which implement reselection this allows the target to reconnect to the initiator to continue some operation that was previously started by the initiator and suspended by the target. The RESELECTION phase is not implemented on the Macintosh computer.

- **Command phase**

The COMMAND phase allows the target to request command information from the initiator. The command information instructs the target what function it is expected to complete.

- **DATA phase**

The DATA phase allows the transfer of parameters or data from the target to the initiator or from the initiator to the target.

- **STATUS phase**

The STATUS phase allows the target to request that transfer status information be sent from the target to the initiator.

- **Message phase**

The MESSAGE phase allows message information to be sent from the target to the initiator or from the initiator to the target. Multiple messages may be transferred.

A completed SCSI operation will start with a BUS FREE phase and must proceed through an ARBITRATION phase, SELECTION phase, COMMAND phase, STATUS phase, and a MESSAGE phase. In addition to this the SCSI operation may include a RESELECTION phase and a DATA phase. This sequence can only be broken through a time-out or the undesirable assertion of the bus RESET signal at which time the bus must be released to the BUS FREE phase.

The Macintosh does not support the RESELECTION phase.

1.2 SCSI Bus Signalling

The SCSI bus consists of 18 signal lines, nine of which are control signal lines and nine of which are data signal lines.

During a sequence of bus phases the bus control signals are asserted in a complicated control and handshaking sequence. The sequence the control signals may be asserted is specified in the ANSI standard. A typical SCSI transfer is discussed in section 5.2.4. Minimal and maximal duration between signal transitions is also specified in the standard. The SCSI bus signals are described below (all signals are active low):

- **BSY (Busy)** : An 'or-tied' signal that indicates the bus is being used.

- **SEL (Select)** : A signal used by an initiator to select a target or by a target to reselect an initiator.
- **C/D (Command/Data)** : A signal driven by the target that indicates control or data information is on the bus. True (active low) indicates control data.
- **I/O (Input/Output)** : A signal driven by the target that controls the direction of data movement on the data bus with respect to the initiator. True indicates input to the initiator.
- **MSG (Message)** : A signal driven by the target during the message phase.
- **REQ (Request)** : A signal driven by a target to indicate the request for a REQ/ACK data transfer handshake.
- **ACK (Acknowledge)** : A signal driven by an initiator to indicate an acknowledgement for a REQ/ACK data transfer handshake.
- **ATN (Attention)** : A signal driven by an initiator to indicate the attention condition.
- **RST (Reset)** : An 'or-tied' signal that indicates the reset condition.
- **DB(7-0,P)** : Eight data bit signals, plus a parity bit signal that form the data bus.

Certain SCSI bus signals are driven only by the initiator or only by the target. Others are driven either by the initiator or by the target depending on the bus phase. The following table lists all the SCSI bus signals (except RST) and their relationship to the bus phases. RST can be driven by any device but is completely asynchronous and is not constrained to any bus phases.

BUS PHASE	SIGNALS AND THEIR DRIVE SOURCES						
	$\overline{\text{BSY}}$	$\overline{\text{SEL}}$	$\overline{\text{SD0-SD7, SDP}}$	$\overline{\text{I/O}}$	$\overline{\text{C/D, MSG, REQ}}$	$\overline{\text{ATN}}$	$\overline{\text{ACK}}$
Bus free	None	None	None	None	None	None	None
Arbitration	All	Winner	ID bit	None	None	None	None
Selection	Initiator, Target	Initiator	Initiator	None	None	Initiator	None
Reselection	Initiator, Target	Target	Target	Target	None	None	None
Data out	Target	None	Initiator	Target	Target	Initiator	Initiator
Data in	Target	None	Target	Target	Target	Initiator	Initiator
Command out	Target	None	Initiator	Target	Target	Initiator	Initiator
Status in	Target	None	Target	Target	Target	Initiator	Initiator
Message out	Target	None	Initiator	Target	Target	Initiator	Initiator
Message in	Target	None	Target	Target	Target	Initiator	Initiator

Fig I.1 - SCSI Signal Sources (SBC Data Manual, 1990)

1.3 General SCSI Commands

As discussed in section 4.4.1.2 the completion of a SCSI command involves the transfer of a command descriptor block from the **initiator** to the **target**. In order for a device to adhere to the SCSI specification a number of general commands must be implemented. Out of 256 available commands four commands are classed as mandatory, four commands are for devices that support independent self configuring software, twenty two commands are optional, twenty three commands are vendor specific, with the rest reserved for future use.

The classification of commands as mandatory or optional is dependant on the device type. Device types include direct access devices, sequential access devices, printer devices, processor devices and WORM devices. The list below summarises important commands for processor devices, direct access device and commands common to all device types:

Op Code	Type	Command name
Group 0 commands common to all device types :		
00h	Optional	Test Unit Ready
03h	Mandatory	Request Sense
12h	Self Conf. SW	Inquiry
18h	Optional	Copy
1Ch	Optional	Receive Diagnostic Results
1Dh	Optional	Send Diagnostic
Group 1 commands common to all device types :		
39h	Optional	Compare
3Ah	Optional	Copy and Verify
Group 0 commands for Processor Devices :		
08h	Optional	Receive
0Ah	Mandatory	Send
Group 0 commands for Direct-Access Devices :		
01h	Optional	Rezero Unit
04h	Mandatory	Format Unit
07h	Optional	Reassign Blocks
08h	Mandatory	Read
0Ah	Mandatory	Write

0Bh	Optional	Seek
15h	Optional	Mode Select
16h	Optional	Reserve
17h	Optional	Release
18h	Optional	Copy
1Ah	Optional	Mode Sense
1Bh	Optional	Start/Stop Unit
1Eh	Optional	Prevent/Allow Medium Removal

Appendix J

SCSI Bus Controller Specifications

The SN75C091A SBC manufactured by Texas Instruments is a single ended flexible SCSI implementation for microprocessors. It provides DMA or programmed I/O capabilities and can be interrupt driven to minimise host polling. The SBC consists of a single 68 pin PLCC package. The SN75C091A can execute multiphase commands to minimise host interrupts. Chip access is provided through 32 directly addressable registers (Texas Instruments, 1990).

J.1 SBC Features

SCSI Bus Interface

- Complies with ANSI X3.131-1986 SCSI standard
- Performs INITIATOR and TARGET functions
- Supports arbitration, selection, and reselection
- Performs asynchronous data transfers of up to 5 Megabytes/second (MBps)
- Performs synchronous data transfers of up to 5 Megabytes/second (MBps) with programmable offset up to 15
- Has on-chip 48-mA transceivers
- Provides optional parity generation, checking, and pass-through
- Reduces overhead associated with initiator multi-threading by automatically handling save-data-pointer messages, disconnects, and reconnects
- Performs automatic message and command-length decoding
- Has two 32-byte FIFOs for command and message preloading

Microprocessor Interface

- Provides chip control via directly-addressable registers
- Has optional address latch line for multiplexed address/data buses
- Allows DMA- or programmed-I/O data transfers
- Is interrupt-driven to minimize host polling
- Can execute multi-phase commands to minimize interrupts
- Has 24-bit transfer counter
- Provides byte-stacking control to accommodate 8-, 16-, and 32-bit systems

J.2 Block Diagram

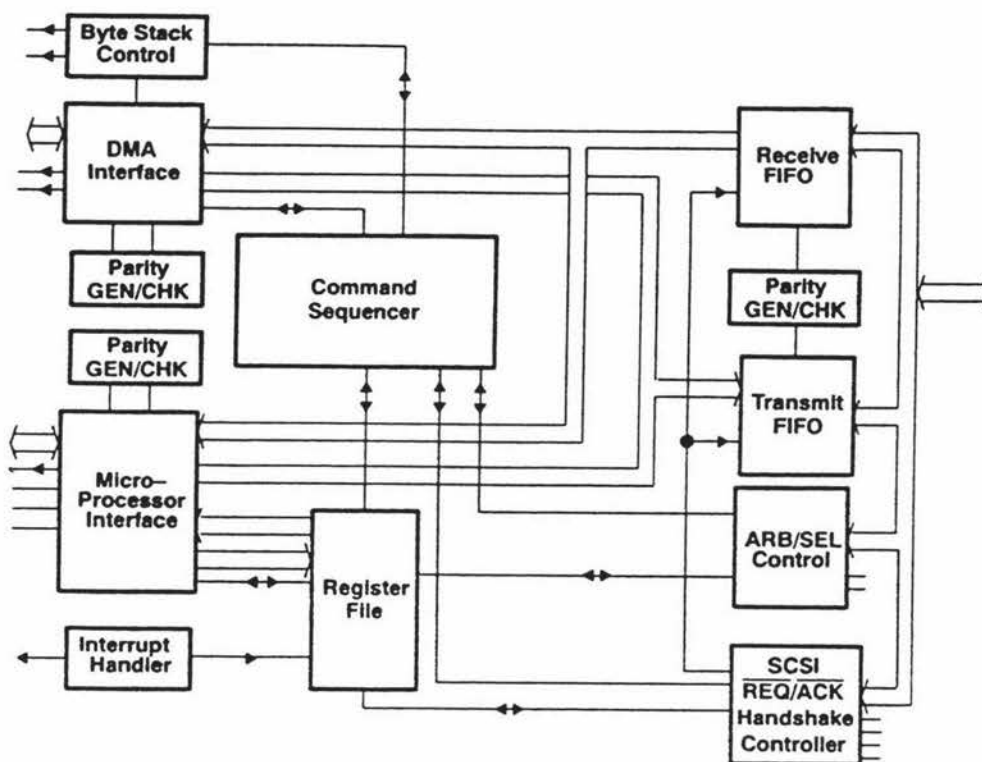


Fig J.1 - SN75C091A Functional Block Architecture

(SBC Data Manual, 1990)

The SBC provides a microprocessor port for information transfer and chip control. A separate DMA port is also provided for SCSI data transfers between memory and the SCSI bus. The DMA port may be connected directly to an 8-bit system or through byte stack registers to 16-, 24-, and 32-bit systems.

J.3 Registers

REGISTER ADDRESSES						
A4	A3	A2	A1	A0	READ/WRITE	REGISTER
0	0	0	0	0	R	Receive FIFO
0	0	0	0	0	W	Transmit FIFO
0	0	0	0	1	R/W	Command
0	0	0	1	0	R	Transfer status
0	0	0	1	1	R	Bus phase status
0	0	1	0	0	R	Function interrupt status
0	0	1	0	1	R	Error interrupt status
0	0	1	1	0	R/W	Interrupt enable

REGISTER ADDRESSES						
A4	A3	A2	A1	A0	READ/WRITE	REGISTER
0	0	1	1	1		(Reserved)
0	1	0	0	0	R/W	Control
0	1	0	0	1	R/W	Byte stack control
0	1	0	1	0	R/W	Parity control
0	1	0	1	1	R/W	Synchronous transfer
0	1	1	0	0	R/W	Selection or Reselection timeout
0	1	1	0	1	R/W	Self-ID
0	1	1	1	0	R/W	Destination ID
0	1	1	1	1	R	Source ID
1	0	0	0	0	R/W	Target LUN
1	0	0	0	1	R/W	Command state
1	0	0	1	0	R/W	Transfer counter (least significant byte)
1	0	0	1	1	R/W	Transfer counter (middle byte)
1	0	1	0	0	R/W	Transfer counter (most significant byte)
1	0	1	0	1	R	Backup counter (least significant byte)
1	0	1	1	0	R	Backup counter (middle byte)
1	0	1	1	1	R	Backup counter (most significant byte)
1	1	0	0	0	R/W	Offset counter
1	1	0	0	1		(Reserved)
1	1	0	1	0	R/W	Test control
1	1	0	1	1	R	Test points register 0
1	1	1	0	0		(Reserved)
1	1	1	0	1		(Reserved)
1	1	1	1	0		(Reserved)
1	1	1	1	1		(Reserved)

- **Transmit & Receive FIFOs**

Two 32-byte transmit and receive registers are used to buffer the SCSI bus information transfers. The Receive and Transmit FIFOs are accessed through the microprocessor port at register file address 00000h. Writing loads a byte into the transmit FIFO through the microprocessor port; reading enables the information onto the microprocessor port and unloads the byte from the receive FIFO.

- **Command Register**

The command register stores the commands written by the microprocessor. Each command is executed immediately upon being sent to the chip. Generally the microprocessor should not issue a new command to the SBC while the previous command is still active.

- **Transfer Status & Bus Phase Status Register**

Registers that contain status bits which reflect the status of the SBC chip and of the SCSI bus.

- **Functional Interrupt Status & Error Interrupt Status Register**

Registers that contain status bits which reflect the status of the SBC functional interrupts and error condition interrupts.

- **Variety of other registers**

Variety of other registers that contain control information, status information, SCSI ID information, and transfer counters.

J.4 SBC Chip Commands

The SBC is driven by chip commands written to the COMMAND register. These commands are instructions from the microcontroller to the SBC to modify the current bus phase or transfer data. These commands fall in three categories:

- Non interrupting commands
- Single phase interrupting commands
- Multiphase interrupting commands

Noninterrupting Commands

COMMAND CODE	COMMAND NAME	ISSUED STATE	RESULT STATE
00000	Chip Reset	ANY	D
00001	Disconnect	T, TO	D
00010	Pause	I, T	I, T
00011	Assert ATN	I	I
00100	Negate ACK	I	I
00101	Clear Receive FIFO	D, I, T	D, I, T
00110	Clear Transmit FIFO	D, I, T	D, I, T

Single-Phase Interrupting Commands

COMMAND CODE	COMMAND NAME	ISSUED STATE	RESULT STATE
00111	SCSI Bus Reset	ANY	D
01000	Select with ATN	D	I
01001	Select without ATN	D	I
01010	Reselect	D	T
01011	(reserved)	—	—
01100	Receive Command	T	T
01101	Receive Data	T	T
01110	Receive Message Out	T	T
01111	Receive Unspecified Information Out	T	T
10000	Send Status	T	T

10001	Send Data	T	T
10010	Send Message In	T	T
10011	Send Unspecified Information In	T	T
10100	Transfer Information	I	I
10101	Transfer Pad	I	I
10110	(reserved)	–	–
10111	(reserved)	–	–

Multiphase Interrupting Commands

COMMAND CODE	COMMAND NAME	ISSUED STATE	RESULT STATE
11000	Select with $\overline{\text{ATN}}$ and Transfer	D, I	D
11001	Select without $\overline{\text{ATN}}$ and Transfer	D	D
11010	Reselect and Receive Data	D	T
11011	Reselect and Send Data	D	T
11100	Wait for Select with $\overline{\text{ATN}}$ and Receive	D, T	T
11101	Wait for Select without $\overline{\text{ATN}}$ and Receive	D, T	T
11110	Conclude	T	D
11111	Link to Next Command	T	T

STATE
D = Disconnected
I = Initiator
T = Target
TO = Time-Out

A normal command sequence for the SBC used in a target role would involve waiting for the chip selection using the an interrupting multiphase command or by directly polling the transfer status register.

Once the chip has been selected a **receive command** command would be sent to receive the SCSI command **command descriptor block**. Based on the information in the command descriptor block additional data transfer command may be executed.

To complete the SCSI transfer a message byte and a status byte need to be sent. This can be completed using the conclude command.

J.5 SBC Electrical Characteristics

Absolute Maximum Ratings Over Free-Air Temperature Range (Unless Otherwise Noted)

Supply voltage range, V_{CC} (see Note 1)	– 0.5V to 7 V
Input voltage range, V_I , at any input	– 0.5V to 7 V
Output voltage range, V_O	– 0.5V to 7 V
Storage temperature range	– 65°C to 150°C
Case temperature for 10 seconds	260°C

NOTE 1: All voltage values are with respect to GND.

Recommended Operating Conditions

	MIN	NOM	MAX	UNIT
Supply voltage, V_{CC}	4.75	5	5.25	V
High-level input voltage, V_{IH}	2		V_{CC}	V
Low-level input voltage, V_{IL}^\dagger	– 0.5		0.8	V
Clock frequency, f_{clock}		20		MHz
Operating free-air temperature, T_A	0		70	°C

† The algebraic convention, in which the least positive (most negative) value is designated minimum, is used in this data manual for logic voltage levels only.

Electrical Characteristics Over Recommended Ranges of Supply Voltage and Operating Free-Air Temperature (Unless Otherwise Noted)

PARAMETER		TEST CONDITIONS	MIN	TYP †	MAX	UNIT
V_{OH} High-level output voltage		$I_{OH} = -4$ mA (see Note 2)	3.7			V
		$I_{OH} = -2$ mA (see Note 3)				
V_{OL} Low-level output voltage		$I_{OL} = 48$ mA (see Note 4)			0.5	V
		$I_{OL} = 4$ mA (see Note 2)				
		$I_{OL} = 2$ mA (see Note 3)				
I_I Input current		$V_{CC} = 5.25$ V, $V_I = 0$ to 5.25 V			± 10	μ A
I_{OZ} High-impedance output current		$V_{CC} = 5.25$ V, $V_I = 0$ to 5.25 V			± 10	μ A
I_{CC} Supply current		No load on outputs, $f = 20$ MHz			30	mA
C_i Input capacitance	Input pins	$V_{CC} = 5$ V, $T_A = 25^\circ\text{C}$		5		pF
	Bidirectional pins			13		pF
C_o Output capacitance	Output pins	$V_{CC} = 5$ V, $T_A = 25^\circ\text{C}$		8		pF

† All typical values are at $V_{CC} = 5$ V and $T_A = 25^\circ\text{C}$.

NOTES: 2. Applies to MP, M(0:7) and DP, D(0:7) only.

3. Applies to all other outputs or bidirectional signals.

4. Applies to SCSI interface signals only.

Appendix K

Macintosh SCSI Manager

The Macintosh SCSI Manager must be used to program the SCSI interface on the Macintosh computer.

K.1 Macintosh SCSI Manager SCSI Calls

The Macintosh SCSI manager provides the following SCSI calls:

- **SCSIGet()**
Arbitrate for the SCSI bus.
- **SCSISelect(*targetID*)**
Select a SCSI device with a specific ID (*targetID*).
- **SCSICmd(*buffer*, *count*)**
Send a command to the selected target device. Where *buffer* is a pointer to a command descriptor block and *count* is the size of the command descriptor block pointed to by *buffer*.
- **SCSIComplete(*stat*, *message*, *wait*)**
Gives the current command a given number of ticks to complete. The status and message bytes returned by the target device are returned in *stat* and *message*. The maximum number of ticks to wait (time-out) is specified in *wait*.
- **SCSIRead(*tibPtr*)**
Transfer data from the target to the initiator, as specified in the transfer instruction block pointed to by *tibPtr*.
- **SCSIWrite(*tibPtr*)**
Transfer data from the initiator to the target, as specified in the transfer instruction block pointed to by *tibPtr*.

- **SCSIRBlind(*tibPtr*)**

Transfer data from the target to the initiator, as specified in the transfer instruction block pointed to by *tibPtr*, without byte handshaking by polling and waiting for the /REQ line after each byte.

- **SCSIWBlind(*tibPtr*)**

Transfer data from the initiator to the target, as specified in the transfer instruction block pointed to by *tibPtr*, without byte handshaking by polling and waiting for the /REQ line after each byte.

- **SCSISelAtn(*targetID*)**

Select a SCSI device and signal the intention to send a message by asserting the ATN line.

- **SCSISat()**

Return a bitmap of the SBC control and status registers.

- **SCSIMsgIn(*message*)**

Get a message from the SCSI device.

- **SCSIMsgOut(*message*)**

Send a message to the SCSI device.

- **SCSIReset()**

Reset the SCSI bus by asserting the RST line.

All SCSI Manager SCSI calls return an error code indicating the success or failure of the function. Error codes are 0 = no error, while any other value indicates a command specific error has occurred.

K.1 Transfer Instruction Blocks (TIBs)

The transfer of data from the target to the initiator or vice versa requires a transfer instruction block (TIB) for the data transfer calls on the Macintosh. A TIB is a sequence of low level instructions that tell the SCSI Manager what to do with the data bytes transferred during the data phase. A TIB contains a pseudo-program consisting of a variable number of instructions which are interpreted by the SCSI Manager. TIB instructions are similar to assembly code but with a very limited instruction set.

Eight instructions are available:

- **scInc *buffer count***

The scInc instruction moves *count* bytes to or from *buffer*, incrementing *buffer* by *count* when done.

- **scNoInc *buffer count***

The scNoInc instruction moves *count* bytes to or from *buffer*, leaving *buffer* unmodified.

- **scAdd *addr value***

The scAdd instruction adds *value* to the address in *addr* (performed as MC68000 addition operation).

- **scMove *addr1 addr2***

The scMove instruction moves the value of the location pointed to by *addr1* to the location pointed to by *addr2* (performed as MC68000 move operation).

- **scLoop *relAddr count***

The scLoop instruction decrements *count* by 1. If the result is greater than 0, the pseudo-program execution resumes at the *current address + relAddr*. If the result is 0, execution resumes at the next instruction. *RelAddr* should be a signed multiple of the instruction size (10 bytes). For example, to loop to the immediately preceding instruction, the *relAddr* field would contain -10.

- **scNop *nil nil***

The scNop instruction does nothing.

- **scStop *nil nil***

The scStop instruction terminates pseudo-program execution, returning to the calling SCSI Manager routine.

- **scComp *addr count***

The scComp instruction may be used for data verification and can be used only with a read command. Beginning at *addr*, it compares incoming data bytes with memory, incrementing *addr* by *count* when done. If the bytes do not compare equally, an error is returned to the SCSI Manager read command.

For example, a TIB to transfer six 512 byte blocks of data from or to address 0x67B50:

scOpcode	scParam1	scParam2
scInc	0x67B50	512
scLoop	-10	6
scStop		

TIBs can read in variable length data blocks by using self modifying code. For example, if the first bytes in a data block sent from another SCSI device contains the length of the data block, these bytes may be read into the second parameter of the next scInc instruction to correctly read in the required number of bytes. This capability is used in the SCSI transfer byte loss detection and resend scheme.

Appendix L

SCSI Byte Loss Detection and Resend Scheme

L.1 Byte Loss Detection and Resend Scheme

The byte loss detection and resend scheme is an elaborate scheme to ensure transferred data is not corrupted by buffer overflow (see section 5.2.4.2). It will correct for occasional buffer overflow of up to several hundred bytes.

The implementation of the byte loss detection scheme required some major algorithm and software changes. The detection and resend scheme works on the basic principle that the SBC expects to send a certain number of bytes. If at the end of the image line the SBC expects to send more bytes, the SBC FIFO must have overflowed during the A/D conversion loop. The line that was sent must be ignored and the line resent from the scanner memory.

Two restrictions that must be taken into account are that:

1. The TIB instruction set is very limited and can only execute seven types of instructions (see APPENDIX K for more detail on TIBs).
2. The scanner SCSI bus controller provides limited status information. There are flags that indicate whether the Transmit FIFO is full or half full, but not whether the transmit FIFO is empty.

The detection and correction scheme works on the following principle:

1. Once all the image bytes for a line have been sent and the FIFO is less than half full, send another 15 filler bytes. Wait until all bytes have been transmitted. If the SBC transfer counter is not equal to zero more than 15 bytes have been lost and the line needs to be resent. Send one more filler byte and wait until it could have been sent. If the SBC FIFO half full flag has not been set, less than 15 bytes have been lost and the line needs to be resent (see point 4).
2. If no bytes were lost the FIFO will now contain 16 filler bytes which must be cleared before the next line is transferred.
3. Now if the transfer of the image line was successful the TIB needs to increment the pointer to memory where the image is stored so the next line can be captured. If however the bytes were lost, the memory will contain invalid information and the next line must be captured to overwrite the invalid information. This is achieved by using the feature that TIBs can contain self modifying code. The microcontroller has determined whether the line is valid. Based on this an increment number is transferred which the TIB uses as the amount by which to increment the memory data pointer. A second number must be transferred which the amount by which to decrement the TIB loop counter to ensure the correct number of lines are captured (see TIB in figure L.2).
4. If however bytes were lost during the line, the line must be resent. This involves sending filler bytes until the correct number of bytes been sent (transfer counter = zero). The scanner SBC is then cleared of filler bytes and a memory increment and TIB loop decrement for an invalid line are sent. This means the last line is disregarded. The line is resent from memory, and the memory increment and TIB loop decrement for a valid line are sent.
5. Now the microcontroller can loop back and capture the next line.

The implementation of this byte loss detection and resend scheme has little impact on the acquisition plug-in code. Only the TIB required additions to allow for significant self modification.

The implementation of this byte loss detection and resend scheme does have implications on the structure of the microcontroller *image block capture* algorithm, and timing of the A/D conversion loop discussed in section 5.2.3. The A/D conversion loop must write the image data to RAM as well as to the SCSI. This increases the A/D conversion loop to 12 μ s as shown in figure 5.9. Secondly the *image block capture*

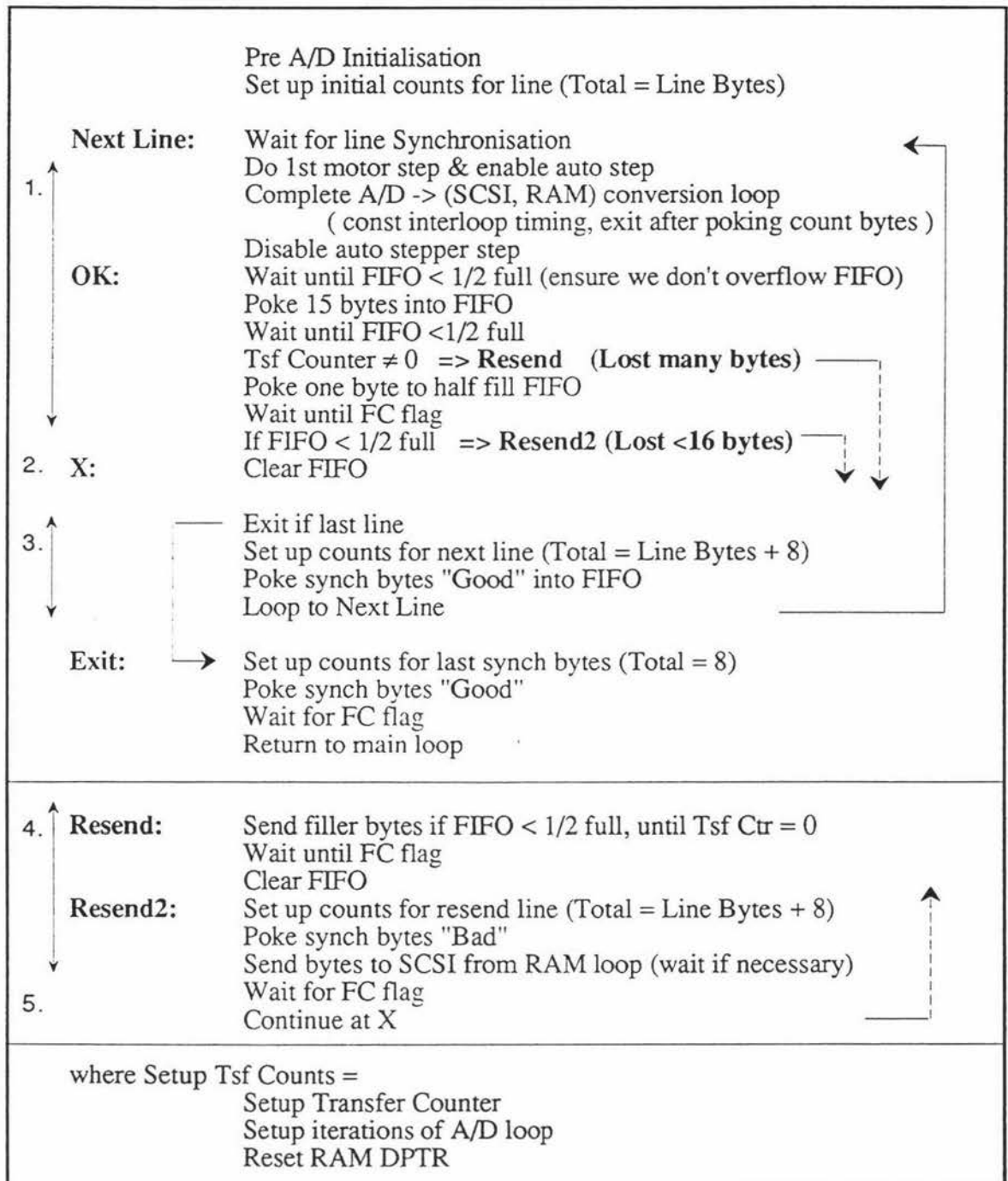


Figure L.1- Byte loss detection and resend scheme

algorithm (figure 5.6) must be modified to accommodate the additional write to RAM, additional end of line checking, and a line resend if necessary. The principle of the Image block capture algorithm does remain the same.

```

myTIB[0].scOpcode = scNoInc;
myTIB[0].scParam1 = bufferPtr;
myTIB[0].scParam2 = 1;

myTIB[1].scOpcode = scNoInc;
myTIB[1].scParam1 = bufferPtr;
myTIB[1].scParam2 = lwidth;

myTIB[2].scOpcode = scNoInc;
myTIB[2].scParam1 = &myTIB[3].scParam2+3;
myTIB[2].scParam2 = 1;

myTIB[3].scOpcode = scNoInc;
myTIB[3].scParam1 = &Dummy;
myTIB[3].scParam2 = 0;

myTIB[4].scOpcode = scNoInc;
myTIB[4].scParam1 = &myTIB[6].scParam2;
myTIB[4].scParam2 = 4;

myTIB[5].scOpcode = scNoInc;
myTIB[5].scParam1 = &myTIB[7].scParam2;
myTIB[5].scParam2 = 4;

myTIB[6].scOpcode = scAdd;
myTIB[6].scParam1 = &myTIB[1].scParam1;
myTIB[6].scParam2 = lwidth;

myTIB[7].scOpcode = scAdd;
myTIB[7].scParam1 = &myTIB[8].scParam2;
myTIB[7].scParam2 = 0;

myTIB[8].scOpcode = scLoop;
myTIB[8].scParam1 = -70;
myTIB[8].scParam2 = Lines;

myTIB[9].scOpcode = scStop;
myTIB[9].scParam1 = nil;
myTIB[9].scParam2 = nil;

```

Annotations:

- Read in one image line (points to myTIB[1])
- Fix for byte gain problem (points to myTIB[2])
- Read buffer memory pointer increment (points to myTIB[4])
- Read TIB loop decrement (points to myTIB[5])
- Increment memory pointer by 0 or 1024 (points to myTIB[6])
- Increment loop counter by 1 or 0 (points to myTIB[7])
- Capture lines until #Lines captured (points to myTIB[8])

Figure L.2 - TIB for byte loss detection and resend scheme

(for source see Appendix M - File Functions.c)

Appendix M

Scanner Control Software

This appendix presents relevant sections of the scanner control software. An overview of the code is provided with a breakdown into files. Listings are also provided of relevant sections of code. This includes the microcontroller assembly code as well as the TreeScan acquire plug-in.

- Microcontroller code (version 2.0)
- TreeScan Acquire Plug-in (version 3.28)

M.1 Microcontroller Code (Version 2.0)

The microcontroller source code is written in assembly language and is divided into seven source files (2800 lines of code) with four extra files containing the 10 bit A/D lookup tables. The main file TASM200.ASM contains important code documentation. This includes microcontroller I/O port declarations, memory map, register usage, variable naming convention, modification history, constant declarations, and variable declarations.

Microcontroller code source files

TASM200.ASM	Main file which #includes all other source files. Contains important code documentation, variable and constant declaration, and main SCSI background loop.
REDEF517.ASM	Register redefinition to allow assembler to assemble for the 80C518A microcontroller.
INIT.ASM	Bootup register and port initialisations.
MACLIB.ASM	Inline macro libraries.
SUBRT1.ASM	Subroutine library 1 : Implementation of most of the SCSI command routines and other subroutines.
SUBRT2.ASM	Subroutine library 2 : Image block capture routine (8-bit).
SUBRT3.ASM	Subroutine library 3 : Image block capture routine (10-bit).
ADTABLES.ASM	10-bit A/D lookup table implementation.
ADTABLE1.ASM	A/D lookup table 1.
ADTABLE2.ASM	A/D lookup table 2.
ADTABLE3.ASM	A/D lookup table 3.

A complete listing of the files implementing interesting and relevant code has been included in this appendix (files highlighted in bold).

M.1.1 TASM200.ASM Source Listing

```

*****
;*
;*   TreeScan Microcontroller Software Version 2.0           ( 1/ 2/95)
;*
;*   Siemens 80C517A with 16 MHz clock
;*
;*   Port based I/O
;*   Port 0           Multiplexed data and low order address bus
;*
;*   Port 1   P1.0           ( )
;*             P1.1   Mirror Stepper clock           (out)
;*             P1.2   Line/Integration               (out)
;*             P1.3           ( )
;*             P1.4           ( )
;*             P1.5   Data rate Control              (out)
;*             P1.6           ( )
;*             P1.7   T2 A/D count input             (in)
;*
;*   Port 2           High order data bus
;*
;*   Port 3   P3.0   RS232 Rx                         (in)
;*             P3.1   RS232 Tx                         (out)
;*             P3.2   Stepper direction               (out)
;*             P3.3   Stepper Mode (full/half)         (out)
;*             P3.4   Focus stepper direction          (out)
;*             P3.5   Focus stepper Clock             (out)
;*             P3.6   Addressing -WR signal           (out)
;*             P3.7   Addressing -RD signal           (out)
;*
;*   Port 4   P4.0           ( )
;*             P4.1   Lens aperture signal (16 level) (out)
;*             P4.2   Pulsed data rate                 (out)
;*             P4.3   LED Output (debug2)              (out)
;*             P4.4   Stepper Pulse signal             (out)
;*             P4.5           ( )
;*             P4.6           ( )
;*             P4.7           ( )
;*
;*   Port 5   P5.0   Lens zoom out                    (out)
;*             P5.1   Lens zoom in                     (out)
;*             P5.2   Lens focus Far                   (out)
;*             P5.3   Lens focus Near                  (out)
;*             P5.4   Home pos 1                       (in )
;*             P5.5   Home pos 2                       (in )
;*             P5.6           ( )
;*             P5.7           ( )
;*
;*   Port 6   P6.0   A/D start trigger                 (in)
;*             Returned Data Synch signal              (in )
;*             P6.1   Line Synch in                   ( )
;*             P6.2           ( )
;*             P6.3           ( )
;*             P6.4           ( )
;*             P6.5           ( )
;*             P6.6           ( )
;*             P6.7           ( )
;*
;*   Port 7   P7.0   A/D video signal in               (in)
;*             P7.1   Lens focus feedback             (in)
;*             P7.2   lens zoom feedback              (in)
;*             P7.3           ( )
;*             P7.4           ( )
;*             P7.5           ( )
;*             P7.6           ( )
;*             P7.7           ( )
;*
;*   Port 8   P8.0           ( )
;*             P8.1           ( )
;*             P8.2           ( )
;*             P8.3           ( )
;*
;*   Memory Mapped I/O
;*   FE00h      SCSI FIFO
;*   FE01h      SCSI Command register
;*   FE02h      SCSI Transfer status reg
;*   to FEFFh   SCSI Further registers
;*
;*   Memory Map
;*   Code Space
;*   0000 to FFFF Program memory (RAM)
;

```

```

;*
;*      Data Space
;*      0000 to FDFE External RAM
;*      FE00 to FEEF SCSI chip memory mapped IO
;*      FF00 to FFFF P2 Regen registers
;*
;* CPU Register bank usage
;* Bank 0
;* Bank 1
;* Bank 2
;* Bank 3
;*
;* Register usage within banks
;* R0
;* R1
;* R2-R3      Counters for inside AD Loop (H & L)
;*             Return values for maths routines (H & L)
;*             Flashing LED counters (H & L)
;* R4-R5      Delay Loop counters (H & L)
;* R6-R7      Temporary Register (H & L)
;*
;* Data Pointer Usage
;* DPTR 0      Miscellaneous SCSI Registers
;* DPTR 1      SCSI FIFO
;* DPTR 2      SCSI Command Register
;* DPTR 3      SCSI Transfer Status Register
;* DPTR 4      Temporary Data Buffer in Memory
;* DPTR 5      DPTR to point to ADTables
;*
;* Variable and Constant Naming Convention
;*
;*      r_XXX      Subroutine name
;*      m_XXX      Macro name
;*
;* Area of application
;* s....      SCSI Related
;* m....      Motor Related
;* c....      Camera Related
;* l....      lens related
;* o....      Top level variable
;*
;* Variable / constant type
;* .v...      Variable
;* .c...      Command to do
;* .x...      Constant
;* .r...      Register
;* .m...      8 bit Mask
;* .ba...     Bit in the accumulator
;* .p...      Pinout Name
;* .s...      Status variable
;* .q...      Parameter passed in parameter block
;*
;* Program Overview
;*
;*****
;*
;* Change History
;*
;* 2- 6-94  MW  Initial Programming ( V1.0 - Marijn Weehuizen )
;*
;* 25- 6-94  MW  Start on V1.2 - Working SCSI loop
;*
;* 6-10-94  MW  Software fully operational for TreeScan Prototype 1
;*
;* 7-11-94  MW  Start on V1.3 - Code standardisation and documentation
;*
;* 8-11-94  MW  Code Mods - Flash LED if idle
;*                - Up Stepper rate to 330 Hz -> 500 Hz
;*                - Implement automatic return from command
;*
;* 25- 1-95  MW  Implemented Stepper focus routines (@250 Hz)
;*
;* 1- 2-95  MW  Fix immediate return, imlement power off after home mechn
;*
;* 23- 5-95  MW  Reverse High/Low state of infinity posn on focus stepper
;*
;*****
SDEBUG
;-----
;
;      Definition of
;
;      Segment Usage
;      Constants
;      Variables

```



```

;           Registers
;           Commands
;           Status Variables
;-----

;           Code and data space segment names

EPROM      SEGMENT CODE           ;name of code space
SCRATCH    SEGMENT DATA          ;name of internal direct data space
BITS       SEGMENT BIT            ;name of bit addressable space
INDRCT     SEGMENT IDATA          ;name of internal indirect data space

;           Data Pointer Definitions

sxDPTRRegs EQU 00h                ;DPSEL to point to SCSI Reg's
sxDPTRFIFO EQU 01h                ;DPSEL to point DPTR to FIFO
sxDPTRCMD  EQU 02h                ;DPSEL to point DPTR to Command Reg
sxDPTRTsStat EQU 03h             ;DPSEL to point DPTR to TsStat Reg
sxDPTRBuffer EQU 04h             ;DPSEL to point DPTR to Line Buffer
sxDPTRTables EQU 05h             ;DPSEL to point DPTR to AD Tables

;           Macintosh SCSI command Definitions

scmResetDev EQU 0Ch
scmResetMech EQU 0Eh
scmDoXSteps EQU 0Fh
scmPower     EQU 10h
scmEnq       EQU 12h
scmGetXLines8b EQU 14h
scmSetCamOn  EQU 16h
scmSetCamOff EQU 17h
scmZoom      EQU 18h
scmFocus     EQU 19h
scmAperture  EQU 1Ah
scmGetXLines10b EQU 1Bh
scmSetADTable EQU 1Ch
scmFocus2    EQU 1Dh
scmFocus2I   EQU 1Eh

;           SCSI Bus Controller Chip Definitions
;
;           SCSI Register Definitions

sIOBase EQU 0FE00H                ;msb                                     1sb

srFIFO EQU sIOBase + 00h          ;32 9bit Reg
srCMD  EQU sIOBase + 01h          ;DMA M/a DDIR CC4 CC3 CC2 CC1 CC0
srTsStat EQU sIOBase + 02h        ;INT RFE RFHF TFF TFHF TC0 OC0 CDACT
srBusStat EQU sIOBase + 03h       ;INIT TARG - ATN MSG C/D I/O SRST
srIntStat EQU sIOBase + 04h       ;SEL BUS ATN FC DIS - RSL ABEND
srIntErrStat EQU sIOBase + 05h    ;PE UMS SRST T-O NVC CNTL NEWLN HALT
srIntEnab EQU sIOBase + 06h      ;- - - - - FCIE AIE MIE
srCTL EQU sIOBase + 08h          ;SE RE HA HPE AAPE HD HAAM ATNDS
srBSCtl EQU sIOBase + 09h        ;DMD - - - WL1 WL0 BOF1 BOF0
srParCTL EQU sIOBase + 0Ah       ;PMPE MPCE MPGE PPCE PPGE SPE SPCE SPGE
srSychTsStat EQU sIOBase + 0Bh   ;TP3 TP2 TP1 TPO OL3 OL2 OL1 OLO
srTimOut EQU sIOBase + 0Ch       ;TO7 ... TO0
srSelfID EQU sIOBase + 0Dh      ;- - - - - ID2 ID1 ID0
srDestID EQU sIOBase + 0Eh      ;- - - - - ID2 ID1 ID0
srSourceID EQU sIOBase + 0Fh    ;- - - - - ID2 ID1 ID0
srTargLUN EQU sIOBase + 10h     ;1 DSCRVR LUNTAR- - TL2 TL1 TL0
srCMDState EQU sIOBase + 11h    ;SDP - - - CS3 CS2 CS1 CS0
srTsStatCtrl EQU sIOBase + 12h  ;\
srTsStatCtrlM EQU sIOBase + 13h ;1 1 Register
srTsStatCtrlH EQU sIOBase + 14h ;/
srBakCtrlL EQU sIOBase + 15h    ;\
srBakCtrlM EQU sIOBase + 16h    ;1 1 Register
srBakCtrlH EQU sIOBase + 17h    ;/
srOffsetCtrl EQU sIOBase + 18h  ;OC7 ... OCO
srTestCtrl EQU sIOBase + 1Ah    ;- - - - - - - Loopback
srTestPnt EQU sIOBase + 1Bh

;           SCSI Command Definitions

scChReset EQU 00000b
scClrTxFIFO EQU 00110b
scRxCMD EQU 01100b
scRxDATA EQU 01101b
scRxMSGOut EQU 01110b
scRxInfOut EQU 01111b
scTxStat EQU 10000b
scTxDATA EQU 10001b
scTxMSGIn EQU 10010b
scTxInfIn EQU 10011b
scConclude EQU 11110b

```

```

;      Additional SCSI definitions

sxSelfID      EQU    4                ;SelfID = 4 on SCSI Bus
sxIntEnab     EQU    110b            ;All interrupt enables
sxCTL         EQU    10000000b        ;Selection enab, resel disab,
                                        ; no halt on ATN, cont on par err,
                                        ; no ATN on par err, no halt on discnct,
                                        ; no hold ATN, ATN not disab.
                                        ;Disable all parity checks
sxParity      EQU    00000000b        ;Set up to make 1062 AD conversions
                                        ; (1062+8) = 4*256 + (26+8)h
sxADCountH    EQU    04h              ;Number of synch bytes at end of line
sxADCountL    EQU    26h + 8h         ;Bytes to send to test whether FIFO
sxSynchBytes  EQU    08h              ; is indeed empty.
sxTestEmptyBytes EQU 0Fh

sbaIntPend    EQU    Acc.7            ;Polled int detection mask
sbaSelInt     EQU    Acc.7            ;Target selection mask
sbaFCInt      EQU    Acc.4            ;Function Complete mask
sbaDDIR       EQU    Acc.5            ;Data Direction Bit
sbaTsfCtrZero EQU    Acc.2
sbaTxFIFOFull EQU    Acc.4            ;SCSI Transmit FIFO full bit
sbaTxFIFOHalf EQU    Acc.3           ;SCSI Transmit FIFO half full bit

spSCSIReset   EQU    P1.0            ;SCSI reset line

sxBuffer      EQU    0000h

;      Motor definitions

mpMotorCLK    EQU    P1.1
mpCWDir       EQU    P3.2
mpMode        EQU    P3.3

mbaPosSens1   EQU    Acc.4
mbaPosSens2   EQU    Acc.5
mxResetForwH  EQU    02h
mxResetForwL  EQU    00h

;      Camera defintions

cxDRatePerSlow EQU    8Ah ; 84h        ;Square wave period
                                        ; 20MC=26.6us = * 12 / 2 = 78h
cxDRateCmpSlow EQU    45h ; 42h        ; 5us = 28h X 1/(f(osc)/2)
cxDRatePerFast EQU    60h              ;Square wave period
cxDRateCmpFast EQU    30h              ; 5us = 28h X 1/(f(osc)/2)
cxADCountH    EQU    04h              ;Set up to make 1062 AD conversions
cxADCountL    EQU    26h              ; 1062 = 4*256 + 26h
cxADCountZH   EQU    0FBh             ; -1062 bytes
cxADCountZL   EQU    0DAh
cxADCON0Init  EQU    00100000b        ;External A/D init ->External trig, ip0
cxADCON1Init  EQU    00000000b        ;External A/D init ->7uS conv time
cxT2CntRld    EQU    00010010b        ;Set T2 to Counter, reload
cxSteps0      EQU    0              ;Steps to take between scan lines
cxSteps1      EQU    1
cxStepDelayH  EQU    00h              ;\Delay to allow stepper to settle
cxStepDelayL  EQU    0C8h             ;/ approx 2 mS - 200 pulses
cxOptIntTimeH EQU    02h              ; Optimal integration time
cxOptIntTimeL EQU    0A0h             ; approx 6 mS

cmoCMSEL0Set  EQU    00000100b        ;Or Mask to set CMSEL0
cmoCMEN0Set   EQU    00000100b        ;Or Mask to set CMEN0
cmaDRateCTcon EQU    11111000b        ;And mask to set CT input freq = f(osc)/2
cmaCC1Disab   EQU    11110011b        ;And mask to disable CC1
cmoCC1Enab    EQU    00001000b        ;Or mask to enable CC1
cmoCC2Enab    EQU    00100000b        ; or mask to enable CC2
cmoCC1_2Enab  EQU    00101000b        ;Or mask to enable CC1 & CC2
cmaT2Stop     EQU    11111100b        ;And mask to stop T2
cmaCC1_2Disab EQU    11000011b        ;And mask to disable CC1 & CC2

crCountH      EQU    R2
crCountL      EQU    R3                ;Registers to use as count in AD Loop

cpLineInteg   EQU    P1.2
cpDataRate    EQU    P4.0

cpT2CountIP   EQU    P1.7

;      Lens definitions

lpLens        EQU    P5
lmoZoomIn     EQU    00000001b        ;Or masks to set appropriate lens
lmoZoomOut    EQU    00000010b        ; focus and zoom control bits
lmoFocusNear  EQU    00000100b
lmoFocusFar   EQU    00001000b
lmoCMSEL1Set  EQU    00000010b        ;Or Mask to set CMSEL1

```

```

lmoCMEN1Set    EQU    00000010b        ;Or Mask to set CMEN1

lmaZoomIn      EQU    11111110b        ;And masks to clear appropriate lens
lmaZoomOut     EQU    11111101b        ; focus and zoom control bits
lmaFocusNear   EQU    11111011b        ;
lmaFocusFar    EQU    11110111b        ;

lmaLensClear   EQU    11110000b        ;And mask to clear all lens zoom / focus

lpCWDDir       EQU    P3.4              ;Lens focus direction control
lpMotorCLK     EQU    P3.5              ;Lens focus clock control

lbaInfinity    EQU    Acc.6             ;Bit to test whether Infinity has been
                                         ; on manual lens focus

;      Miscellaneous definitions

orDelayCountH  EQU    R4                ;Registers to use as count in Delay Loops
orDelayCountL  EQU    R5
opPowerSav     EQU    P1.3
orRetByteH     EQU    R3
orRetByteL     EQU    R2

orLEDH         EQU    R2                ;LED Count High order byte
orLEDL         EQU    R3                ;LED Count Low order byte

;opLED         EQU    P4.3              ;LED pin to be toggled

oxLEDH         EQU    07fh
oxLEDL         EQU    00h

oxReturn       EQU    00h

;      AD Lookup table base address definitions

oTable1_1Base  EQU    4000h
oTable1_2Base  EQU    4100h
oTable1_3Base  EQU    4200h
oTable1_4Base  EQU    4300h
oTable1_5Base  EQU    4400h

oTable2_1Base  EQU    4500h
oTable2_2Base  EQU    4600h
oTable2_3Base  EQU    4700h
oTable2_4Base  EQU    4800h
oTable2_5Base  EQU    4900h

oTable3_1Base  EQU    5000h
oTable3_2Base  EQU    5100h
oTable3_3Base  EQU    5200h
oTable3_4Base  EQU    5300h
oTable3_5Base  EQU    5400h

;      Debug Definitions

spDebug2       EQU    P1.0 ;4.3         ;SCSI reset line

;      Indirect Memory area
RSEG SCRATCH

;      Location of the SCSI command block received

sCmdBlock0:    DS      1
sCmdBlock1:    DS      1
sCmdBlock2:    DS      1
sCmdBlock3:    DS      1
sCmdBlock4:    DS      1
sCmdBlock5:    DS      1

;      Location of current status variables

ssCMDCount:    DS      1
ssErrCount:    DS      1

msPositionH:   DS      1
msPositionL:   DS      1

csCycleTime:   DS      1
csIntegTime:   DS      1

csIntegTimeH:  DS      1
csIntegTimeL:  DS      1

lsPosition:    DS      1
lsAperture:    DS      1

```

```

csTableAddrH: DS    1                      ;Tables must start on page boundaries
;      Additional Variables
cvLineCountH: DS    1                      ;Line count variable to be used in
cvLineCountL: DS    1                      ;r_GetXLines routine
cvCountH:     DS    1                      ;Variables to store adcounts for
cvCountL:     DS    1                      ;current line
dvDeb1:       DS    1
dvDeb2:       DS    1
ovReturn:     DS    1                      ;Immediate return from command
;-----
;      SCSI Command block parameters
;-----
;      GetXLines Command
cqLineCountH EQU    sCmdBlock1
cqLineCountL EQU    sCmdBlock2
cqIntTimeH   EQU    sCmdBlock3
cqIntTimeL   EQU    sCmdBlock4
cqStepperSteps EQU sCmdBlock5
;      ResetMechn Command
mqReturn     EQU    sCmdBlock1
RSEG EPROM
;      ljmp      ByPass
;      ljmp      Init
;-----
;      Hardware coded
;      Version Number
;      Date Created
;      Massey Reference
;-----
;      Version number and date created
Ver:         DB      01                      ;\
              DB      00                      ; | Version 1.00
              DB      00                      ;/
Date:        DB      02                      ;\
              DB      06                      ; | 2 June 1994
              DB      94                      ;/
;-----
;      Include appropriate files
;      Register Redefinitions
;      Macro Library
;      Interrupt Service Routines
;      Subroutine Library
;-----
;      Siemens 80C517A Register Redefinition
;      Due to lack of 517 support all SFR's need to be explicitly
;      defined in the source code.
$include Redef517.asm
;      Macro Definitions
$include MacLib.asm
;      Interrupt service routines
; $include ISR.asm
;      Subroutine Definitions
$include Subrtl.asm

```

```

$include Subrt2.asm
$include Subrt3.asm

;      Lookup table implementation
LTab:   DB      11000000b      ;zero
        DB      11100000b      ;one
        DB      11110000b      ;six

FocusTab: DB      00000000b      ;Start of focus
          DB      00000011b      ;modification 1
          DB      00001111b      ;modification 2
          DB      00111111b      ;modification 3
          DB      11111111b      ;modification 4

MStr:   DB      'gedMarijn  Jun 1994'
MStrLen EQU      19
MAStrLen EQU      100

;-----
;      Initialisation and configuration
;
;      Include Initialisation File
;
;      ie. CPU initialisation
;      SCSI chip register setup
;-----

Init:

$include init.asm

;-----
;      Main program loop
;
;      Software waits to be interrupted by macintosh
;      SCSI Bus phases handled
;      Call command routine based on SCSI command
;      Complete SCSI transaction
;-----

MainLp:
        m_WaitSCSIIntLED      ;Wait for SCSI int, flashing LED
        m_IsIntSel            ;ensure it is a Select

;      Send 'Receive Command' command and wait till completed

        mov  DPTR,#srCMD
        mov  A,#scRxCMD        ;Send 'Receive command' command
        movx @DPTR,A

        m_WaitSCSIInt        ;Wait for interrupt and
        m_IsIntFC            ;ensure it is a Function Complete

;      Move Command Block out of the FIFO

        mov  DPTR,#srFIFO      ;Move command block from FIFO
        movx A,@DPTR
        mov  sCmdBlock0,A
        movx A,@DPTR
        mov  sCmdBlock1,A
        movx A,@DPTR
        mov  sCmdBlock2,A
        movx A,@DPTR
        mov  sCmdBlock3,A
        movx A,@DPTR
        mov  sCmdBlock4,A
        movx A,@DPTR
        mov  sCmdBlock5,A

;      Receive Command and select action based on command

        mov  A,sCmdBlock0

        cjne A,#scmEnq,J3      ;Is command enquiry?
        lcall r_Inquiry
        ljmp Conclude

J3:      cjne A,#scmGetXLines8b,J5 ;Is command 8 bit get X Lines?
        lcall r_GetXLines8b
        ljmp Conclude

```

```

J5:      cjne A,#scmSetCamOn,J6      ;Is command Set Camera On?
        lcall r_SetCamOn
        ljmp Conclude

J6:      cjne A,#scmSetCamOff,J7     ;Is command Set Camera Off?
        lcall r_SetCamOff
        ljmp Conclude

J7:      cjne A,#scmDoXSteps,J8      ;Is command Do X Steps?
        lcall r_DoXSteps
        ljmp Conclude

J8:      cjne A,#scmResetMech,J9     ;Is command Reset Mechanism?
        lcall r_ResetMechn
        mov A,ovReturn
        cjne A,#oxReturn,RetCtl
        ljmp Conclude

J9:      cjne A,#scmFocus,J10        ;Is command Focus (motor lens)?
        lcall r_Focus
        ljmp Conclude

J10:     cjne A,#scmZoom,J11         ;Is command Zoom (motor lens)?
        lcall r_Zoom
        ljmp Conclude

J11:     cjne A,#scmAperture,J12     ;Is command Set Aperture
        lcall r_Aperture              ; (motor lens)?
        ljmp Conclude

J12:     cjne A,#scmResetDev,J13     ;Is command complete reset?
        lcall r_ResetDev
        ljmp Conclude

J13:     cjne A,#scmPower,J14        ;Is command Power on / off?
        lcall r_Power
        ljmp Conclude

J14:     cjne A,#scmGetXLines10b,J15 ;Is command 10 bit get X lines?
        lcall r3_GetXLines10b
        ljmp Conclude

J15:     cjne A,#scmSetADTable,J16   ;Is command set A/D table?
        lcall r_SetADTable
        ljmp Conclude

J16:     cjne A,#scmFocus2,J17       ;Is command Focus? (stepper motor)
        lcall r_Focus2
        ljmp Conclude

J17:     cjne A,#scmFocus2I,JF       ;Is command Focus at infinity?
        lcall r_Focus2I              ; (stepper motor)
        ljmp Conclude

JF:      LJMP ErrorTrap

```

```

;      Return Status, wait for an interrupt and check it is a function
;      complete interrupt.

```

```
Conclude:
```

```

    m_Conclude 00h
    m_WaitSCSIInt
    m_IsIntFC

```

```
;      Intercommand actions
```

```
RetCtl:
```

```

    mov ovReturn,#00      ;Clear immediate command return
    inc ssCMDCount        ;Increment the command counter

    ljmp MainLp           ;Loop back to get another command

```

```

;-----
;      Error trap and handling
;-----

```

```
ErrorTrap:  inc ssErrCount      ;Increment the error count
```

```
            jmp Conclude
```

```

            setb spDebug2
            clr  spDebug2
            setb spDebug2

```

```

;-----

```

```
;      Main loop bypass - for code testing purposes
;
;-----
ByPass:      cpl P4.3 ;opLED

              ljmp ByPass

;-----
;      Include three versions of the 10 bit A/D lookup table
;
;      Table1 - Linear range over 100 % of input (equiv to 8 bit)
;      Table2 - Linear range over 25 % of input
;      Table3 - Non linear function over full range
;-----

$include ADTable1.asm
$include ADTable2.asm
$include ADTable3.asm
```


M.1.2 SUBRT2.ASM Source Listing

```

;*****
;* Subroutine Library 2 for TreeScan Microcontroller Software *
;* *
;* RTName Description of what the subroutine does *
;* *
;* r_GetXLines8b 8 bit get block of x lines Command *
;* *
;*****
;-----
;
; Act on 8 Bit SCSI Get X Lines command
;
; The desired number of lines are captured from the camera and sent
; to the Mac via the SCSI port
;-----
; Command block calling parameters :
; No of lines H,L
; ( Int Pulses H,L )
; Stepper steps (0, 1 or 2)
;
; SCSI Info sent to Mac :
;-----

r_GetXLines8b:
; React to parameters passed in command block
;
; mov cvLineCountL,cqLineCountL ;Set up the number of lines
; mov cvLineCountH,cqLineCountH ;to capture
; inc cvLineCountH
; lcall r_SetInteg ;Set integration time
;
; Set data rate clock to fast
; m_SetDataRateClkFast
;
; Set stepper motor direction
; clr mpCWDDir
;
; Do any A/D initialisation
;
; mov ADCON0,#cxADCON0Init ;Select channel 0 for analog input,
; mov ADCON1,#cxADCON1Init ;external trigger
;
; Set up SCSI data phase
; m_SCsICmdToDataPhase ;Send null data Tsf Command
;
; Setup initial counters for next line
; m_SetupTsfCtrlstLine
;
; mov crCountH,#cxADCountH ;\
; inc crCountH ;|Set up iterations of AD Loop
; mov crCountL,#cxADCountL ;/
;
; mov DPSEL,#sxDPTRBuffer ;Initialise DPTR to point to
; mov DPTR,#sxBuffer ;temporary buffer
;
; Wait for line synchronisation
r_L8_Synch:
; clr TF2
; m_WaitTF2 ;Wait for start of line
;
; Depending on the number of steps byte passed
; Manually complete 1st step of stepper motor if necessary
;
; mov A,cqStepperSteps
; cjne A,#cxSteps0,r_L8_J1
; jmp r_L8_J3
r_L8_J1: m_ManualStepperStep
;
; Restart auto stepper step
;
; mov A,cqStepperSteps
; cjne A,#cxSteps1,r_L8_J2
; jmp r_L8_J4

```

```

r_L8_J2:    orl CCEN,#cmoCC1Enab          ;Set up auto motor stepping again
            jmp r_L8_ad

r_L8_J3:    nop                          ;\
            nop                          ;|
            nop                          ;| Wait for m.ManualStepper Step
            nop                          ;| duration if not required
            nop                          ;|
            nop                          ;|
            nop                          ;|
            nop                          ;|
            nop                          ;/

            nop                          ;\ mov A,cqStepperSteps
            nop                          ;/
            nop                          ;\ cjne A,cxZeroSteps,r_L8_J4
            nop                          ;/

r_L8_J4:    nop                          ;\ orl CCEN,#cmoCC1Enab
            nop                          ;/

            nop                          ;\ jmp r_L8_ad
            nop                          ;/

; *****
; Get into loop capturing data and sending it to SCSI
; *****
r_L8_ad:    nop                          ;1
            nop                          ;1
r_L8_ad2:    jb BSY,r_L8_ad2              ;Wait until AD complete ;2
            mov A,ADDPATH                ;read data out ;2
            mov DPSEL,#sxDPTRFIFO        ;1
            movx @DPTR,A                 ;move data to SCSI ;2

            mov DPSEL,#sxDPTRBuffer      ;1
            movx @DPTR,A                 ;move data to SCSI ;2
            inc DPTR                     ;1

            djnz crCountL,r_L8_ad         ;2
            djnz crCountH,r_L8_ad2       ;---
                                           ;16 +4

; Stop automatic stepper step
            anl CCEN,#cmaCC1Disab
            clr mpMotorCLK

; Wait until FIFO empty
            m_WaitFIFOHFull              ;Wait until FIFO < Half Full

            mov DPSEL,#sxDPTRFIFO        ;\
            mov R6,#sxTestEmptyBytes    ;| Send 15 bytes
            mov A,#02h                   ;|
r_L8_L1:    movx @DPTR,A                 ;|
            djnz R6,r_L8_L1              ;/

            m_WaitFIFOHFull              ;Wait until FIFO < Half Full

; Jump to Resend if we have lost any bytes
            mov DPSEL,#sxDPTRTSfStat     ;\
            movx A,@DPTR                 ;| If Tsf Ctr not zero, many
            jb sbaTsfCtrZero,r_L8_J5     ;|
            jmp r_L8_Resend              ;/ bytes have been lost -> resend

r_L8_J5:    mov DPSEL,#sxDPTRFIFO        ;\
            mov A,#02h                   ;| Should half fill FIFO if
            movx @DPTR,A                 ;| everything OK

            m_WaitSCSIInt                ;Wait for interrupt and
            m_IsIntFC                    ;ensure it is a Funtion Complete

            mov DPSEL,#sxDPTRTSfStat     ;\
            movx A,@DPTR                 ;| If FIFO not 1/2 full lost
            jb sbaTxFIFOHFull,r_L8_Cont  ;|
            jmp r_L8_Resend2             ;/ < 15 bytes, Resend line

r_L8_Cont:  m_SCISICmdClearFIFO          ;Clear additional 15+1 bytes
                                           ;in FIFO

            m_SCISICmdToDataPhase        ;Send dummy byte
            m_SCISICmdToDataPhase        ;Send dummy byte

; Decrement line counter and Exit if Last line
            djnz cvLineCountL,r_L8_J6
            djnz cvLineCountH,r_L8_J6
            jmp r_L8_Exit

```

```

r_L8_J6:
; Setup counters for next line
    m_SetupTsfCtr4Line
    mov crCountH,#cxADCountH      ;\
    inc crCountH                  ;|Set up iterations of AD Loop
    mov crCountL,#cxADCountL      ;/

    mov DPSEL,#sxDPTRBuffer        ;Initialise DPTR to point to
    mov DPTR,#sxBuffer             ;temporary buffer

; Send Synchronisation bytes "Good"
;
    m_SendSynchBytes cxADCountH, cxADCountL, 00, 00
    m_SendSynchBytesN cxADCountZH, cxADCountZL, 00, 00

; Jump Back for another line
    ljmp r_L8_Synch

; *****
; Resend line from RAM to SCSI and wait for end of line
; *****
r_L8_Resend:
; Send filler bytes if any required
    m_WaitFIFOHFull                ;Wait until FIFO < Half Full

    mov DPSEL,#sxDPTRFIFO          ;\
    mov A,#02h                    ;| Send single fill byte
    movx @DPTR,A                   ;/

    mov DPSEL,#sxDPTRTsfStat       ;\
    movx A,@DPTR                   ;| If Tsf Ctr not zero, send
    jnb sbaTsfCtrZero,r_L8_Resend ; / filler byte and resend line

    m_WaitSCSIInt                  ;Wait for interrupt and
    m_IsIntFC                      ;ensure it is a Funtion Complete

r_L8_Resend2:
    m_SCSICmdClearFIFO             ;Clear FIFO

    m_SCSICmdToDataPhase           ;Send dummy byte
    m_SCSICmdToDataPhase           ;Send dummy byte

; Setup counters for resend line
    m_SetupTsfCtr4Line
    mov crCountH,#cxADCountH      ;\
    inc crCountH                  ;|Set up iterations of Resend Loop
    mov crCountL,#cxADCountL      ;/

    mov DPSEL,#sxDPTRBuffer        ;Initialise DPTR to point to
    mov DPTR,#sxBuffer             ;temporary buffer

; Send Synchronisation bytes "Bad"
    m_SendSynchBytes 00, 00, 00, 01

; Get into loop sending data to SCSI from RAM
r_L8_ad3:    mov DPSEL,#sxDPTRBuffer
    movx A,@DPTR                    ;move data from RAM
    inc DPTR

    mov DPSEL,#sxDPTRFIFO          ;\
    movx @DPTR,A                   ;move data to SCSI

    mov DPSEL,#sxDPTRTsfStat
    movx A,@DPTR
r_L8_L2:    jb sbaTxFIFOHFull,r_L8_L2

    djnz crCountL,r_L8_ad3
    djnz crCountH,r_L8_ad3

; Wait until finished
    m_WaitSCSIInt                  ;Wait for interrupt and
    m_IsIntFC                      ;ensure it is a Funtion Complete

; Jump to security check whether line sent OK
    jmp r_L8_Cont

```

```

; *****
; Conclude last line and finish command
; *****

r_L8_Exit:
; Setup counter for synch bytes of last line

    mov DPSEL,#sxDPTRRegs
    mov DPTR,#srTsfCtrl
    mov A,#sxSynchBytes          ;Set transfer counter to bytes
    movx @DPTR,A                ;to send

    mov DPSEL,#sxDPTRCMD
    mov A,#scTxDATA              ;Return x bytes command
    movx @DPTR,A

; Send Synchronisation bytes "Good"

;     m_SendSynchBytes cxADCountH, cxADCountL, 00, 00
;     m_SendSynchBytesN cxADCountZH, cxADCountZL, 00, 00

; Stop automatic stepper step
; (Leave line rate going)

    andl CCEN,#cmaCC1Disab

; Return data rate to slow speed

    m_SetDataRateClkSlow

; Wait until SCSI transfer is complete

    m_WaitSCSIInt                ;Wait for interrupt &
    m_IsIntFC                    ;check whether it is FC

    ret

```

M.1.3 SUBRT1.ASM Source Listing

```

;*****
;*      Subroutine Library 1 for TreeScan Microcontroller Software      *
;*      *                                                                *
;*      RName          Description of what the subroutine does        *
;*      *                                                                *
;*      r_Wait1mS      Routine to pause 1 mS                          *
;*      r_Wait2mS      Routine to pause 2 mS                          *
;*      r_Wait10mS     Routine to pause 10 mS                         *
;*      r_Wait40mS     Routine to pause 40 mS                         *
;*      r_Wait80mS     Routine to pause 80 mS                         *
;*      r_Inquiry      Enquiry command                               *
;*      r_Autofocus     Complete autofocus at micro level             *
;*      r_ResetMech     Move mechanism to home position               *
;*      r_SetCamOn      Turn camera signals on                        *
;*      r_SetCamOff     Turn Camera signals off                      *
;*      r_DoXSteps      Move stepper motor X steps                   *
;*      r_Focus         Move lens focus controls                     *
;*      r_Zoom          Move lens zoom controls                     *
;*      r_Aperture      Set lens aperture signal                     *
;*      r_Focus2        Act on SCSI Focus using stepper motor        *
;*      r_Focus2I       Focus at infinity using stepper motor        *
;*      r_ResetDev      Complete reset of device                     *
;*      r_Power         Turn power on / off                           *
;*      r_SetInteg      Routine to set integration time based on Cmdblk3 & 4 *
;*      r_SetADTable    Set A/D lookup table                          *
;*      *                                                                *
;*****

;-----
;      Routine to pause 1 mS
;-----
orDelayCountH EQU R4          ;Registers to use as count in Delay
Loops
orDelayCountL EQU R5

r_Wait1mS:    mov orDelayCountH,#3h
              mov orDelayCountL,#99h
r_Wait1_L1:   djnz orDelayCountL,r_Wait1_L1
              djnz orDelayCountH,r_Wait1_L1

              ret

;-----
;      Routine to pause 2 mS
;-----
r_Wait2mS:    mov orDelayCountH,#6h
              mov orDelayCountL,#33h
r_Wait2_L1:   djnz orDelayCountL,r_Wait2_L1
              djnz orDelayCountH,r_Wait2_L1

              ret

;-----
;      Routine to pause 10 mS
;-----
r_Wait10mS:   mov orDelayCountH,#1Ah
              mov orDelayCountL,#4Ah
r_Wait10_L1:  djnz orDelayCountL,r_Wait10_L1
              djnz orDelayCountH,r_Wait10_L1

              ret

;-----
;      Routine to pause 40 mS
;-----
r_Wait40mS:   mov orDelayCountH,#69h
              mov orDelayCountL,#29h
r_Wait40_L1:  djnz orDelayCountL,r_Wait40_L1
              djnz orDelayCountH,r_Wait40_L1

              ret

;-----
;      Routine to pause 80 mS
;-----
r_Wait80mS:   mov orDelayCountH,#0D2h

```

```

        mov orDelayCountL,#52h
r_Wait80_L1: djnz orDelayCountL,r_Wait80_L1
              djnz orDelayCountH,r_Wait80_L1

              ret

```

```

;-----
;      Act on SCSI Inquiry command
;
;      Information is sent back to the enquirer as to what this
;      device is.
;-----

```

```

;      Command block calling parameters :
;      none
;
;      SCSI Info sent to Mac :
;      5 bytes + MStrLen(19) bytes
;-----

```

```

r_Inquiry:  mov DPSEL,#sxDPTRRegs
            mov DPTR,#srTsfCtrl
            mov A,#5+MStrLen          ;Return x bytes
            movx @DPTR,A

            mov DPSEL,#sxDPTRCMD
            mov A,#scTxDATA           ;Return x bytes command
            movx @DPTR,A

            mov DPSEL,#sxDPTRFIFO
            mov A,#03                 ;Processor device
            movx @DPTR,A

            mov A,ssCMDCount          ;\
            movx @DPTR,A              ;| Misc bytes
            mov A,ssErrCount          ;|
            movx @DPTR,A              ;|
            mov A,#00                 ;/
            movx @DPTR,A

            mov A,#MStrLen            ;3 Additional bytes
            movx @DPTR,A

```

```

r_Inq_L1:  mov R0,#0
            mov DPTR,#MStr
            mov A,R0
            movc A,@A+DPTR            ;\

            mov DPTR,#srFIFO
            movx @DPTR,A
            inc R0
            cjne R0,#MStrLen,r_Inq_L1

```

```

m_WaitSCSIInt      ;Wait for interrupt &
m_IsIntFC          ;check whether it is FC

              ret

```

```

;-----
;      Act on SCSI Autofocus command
;
;      An autofocus porcedure is completed based on the parameters sent in
;      the command block.
;-----

```

```

;      Command block calling parameters :
;      none
;
;      SCSI Info sent to Mac :
;
;-----

```

```

r_Autofocus:

;      Capture the current line into memory

;      Process the line

;      Make focus modification

;      Capture the next line into memory

;      Process the line

```

```

;      If better, loop back to focus modification

;      Return to best focus position

ret

;-----
;      Act on SCSI Reset Mechanism command
;
;      The stepper motor is driven back to it's home position based on the
;      input from its position sensors.
;-----
;      Command block calling parameters :
;      Byte 1 ( 0 = Keep control, Else return immediately)
;      Byte 2 ( 0 = No effect, 1 = Turn power off)
;
;      SCSI Info sent to Mac :
;-----

r_ResetMechn:
;      Respond to command block parameters
;      Return SCSI ctl immediately if required

        mov  A,mqReturn
        mov  ovReturn,A
        cjne A,#oxReturn,r_RstMch_J2
        jmp  r_RstMch_J1

;      Return control immediately
r_RstMch_J2: clr  P4.3

        m_Conclude 00h
        m_WaitSCSIInt
        m_IsIntFC

;      Actually go home
r_RstMch_J1: m_MotorForwards

;      Drive mechanism forward x steps to get away from home position there
r_RstMch_L1: mov  R2,#mxResetForwH
        inc  R2
        mov  R3,#mxResetForwL

r_RstMch_L2: m_MotorStep_3mS

        djnz R3,r_RstMch_L2
        djnz R2,r_RstMch_L2

        mov  A,P5
        jnb mbaPosSens1,r_RstMch_L1      ; If low still at home pos

;      Drive mechanism back until first sensor input is reached

        m_MotorReverse

        m_MotorStep_3mS

r_RstMch_L3: m_MotorStep_2mS

        mov  A,P5
        jnb mbaPosSens1,r_RstMch_L3      ; If high, not home yet

;      continue for x steps past the sensor

;      Drive mechanism forwards until second position sensor is reached
;      and mechanism is home without hysteresis.

        m_MotorForwards

r_RstMch_L4: m_MotorStep_3mS

        mov  A,P5
        jnb mbaPosSens2,r_RstMch_L4      ; If high, not home yet

```



```

;      Check whether power needs to be turned off as well
;      Byte 2 ( 0 = No effect, 1 = Turn power off)

      mov A,sCmdBlock2
      cjne A,#01,r_RstMch_J3
      setb opPowerSav

r_RstMch_J3:
      ret

;-----
;      Act on SCSI Set Camera On command
;-----
;      Command block calling parameters :
;      none
;-----
;      SCSI Info sent to Mac :
;      none
;-----

r_setCamOn:

;      Set data rate clocking

      m_SetDataRateClkSlow

;      Do integration time / Stepper Pulse 2 Initialisation

      mov TH2,#0FBh          ;Set initial T2 value
      mov TL2,#0C7h

      mov CRCH,#0FBh         ;Set T2 reload
      mov CRCL,#0C7h

      mov CCH1,#0FCh         ;Set Delay B4 2nd step @ 200 pulses
      mov CCL1,#08Fh

      mov CCH2,#0FDh         ;Set Delay B4 int time @ 400 pulses
      mov CCL2,#057h

      mov T2CON,#cxT2CntRld   ;set T2 as counter,auto_reload, mode
0
      orl CCEN,#cmaCC2Enab ;cmaCC1_2Enab          ;Set up CC1 & CC2 on T2

      ret

;-----
;      Act on SCSI Set Camera Off command
;-----
;      Command block calling parameters :
;      none
;-----
;      SCSI Info sent to Mac :
;      none
;-----

r_SetCamOff:

;      Stop automatic stepping and Line rate pulse

      anl T2CON,#cmaT2Stop    ;set T2 as counter,auto_reload, mode
0
      anl CCEN,#cmaCC1_2Disab ;Set up CC1 & CC2 on T2

;      Wait until SCSI transfer is complete

      ret

;-----
;      Act on SCSI do X Steps command
;-----
;      The stepper motor is driven back to it's home position based on the
;      input from its position sensors.
;-----
;      Command block calling parameters :
;      Byte 2 dir (bit 1 only)
;      Byte 3&4 Number of steps H & L
;-----

```

```

;      SCSI Info sent to Mac :
;
;-----

r_DoXSteps:
;      Check direction
;
;      mov A,sCmdBlock2
;      jb Acc.0,r_XStep_J1
;      m_MotorReverse
;      jmp r_XStep_J2
r_XStep_J1: m_MotorForwards
r_XStep_J2:
;      Set up Steps in byte R2, R3
;
;      mov R2,sCmdBlock3
;      mov R3,sCmdBlock4
;
;      Do appropriate number of stepper steps
r_XStep_L1: cjne R3,#00,r_XStep_L1b
;      cjne R2,#00,r_XStep_L1a
;      ret
r_XStep_L1a: dec R2
r_XStep_L1b: dec R3
;      jmp r_XStep_L1c
r_XStep_L1c: setb mpMotorCLK ;Do Fast stepping
;      lcall r_wait1mS ; 500 Hz
;      clr mpMotorCLK
;      lcall r_wait1mS
;      jmp r_XStep_L1
;-----
;      Act on SCSI Focus Command
;
;      The lens output is generated focussed near or far.
;-----
;      Command block calling parameters :
;      Byte 1
;      Byte 2 dir (bit 1 only)
;      Byte 3&4 Number of steps H & L
;
;      SCSI Info sent to Mac :
;
;-----

r_Focus:
;      Set up number of steps
;
;      mov R2,sCmdBlock3
;      inc R2
;      mov R3,sCmdBlock4
;
;      Check direction
;
;      mov A,sCmdBlock2
;      jnb Acc.0,r_Focus_Near
;
;      Complete Focus Far Step
r_Focus_Far:
;      orl P5,#lmoFocusFar
;      lcall r_Wait10mS
;      anl P5,#lmaFocusFar
;
;      djnz R3,r_Focus_Far
;      djnz R2,r_Focus_Far
;      jmp r_Focus_J1
;
;      Complete Focus Near Step
r_Focus_Near:
;      orl P5,#lmoFocusNear
;      lcall r_Wait10mS
;      anl P5,#lmaFocusNear
;
;      djnz R3,r_Focus_Near
;      djnz R2,r_Focus_Near
;
;      jmp r_Focus_J1
r_Focus_J1:
;      ret

```

```

;-----
;       Act on SCSI Zoom Command
;
;       The lens output is generated zoomed in or out.
;-----
;       Command block calling parameters :
;           Byte 1
;           Byte 2 dir (bit 1 only)
;           Byte 3&4 Number of steps H & L
;
;       SCSI Info sent to Mac :
;-----

r_Zoom:
;       Set up number of steps

        mov R2,sCmdBlock3
        inc R2
        mov R3,sCmdBlock4

;       Check direction

        mov A,sCmdBlock2
        jnb Acc.0,r_Zoom_In

;       Complete Focus Far Step
r_Zoom_Out:
        orl P5,#lmoZoomOut
        lcall r_Wait40mS
        lcall r_Wait40mS
        lcall r_Wait40mS
        anl P5,#lmaZoomOut

        djnz R3,r_Zoom_Out
        djnz R2,r_Zoom_Out
        jmp r_Zoom_J1

;       Complete Focus Near Step
r_Zoom_In:
        orl P5,#lmoZoomIn
        lcall r_Wait40mS
        lcall r_Wait40mS
        lcall r_Wait40mS
        anl P5,#lmaZoomIn

        djnz R3,r_Zoom_In
        djnz R2,r_Zoom_In

        jmp r_Zoom_J1

r_Zoom_J1:
        ret

;-----
;       Act on SCSI Aperture Adjust Command
;
;       The lens output is generated zoomed in or out.
;-----
;       Command block calling parameters :
;           Byte 1   No of pulses of duty cycle high
;
;       SCSI Info sent to Mac :
;-----

r_Aperture:

        mov cmh1,#0FFh                ;Set compare value for aperture
        clr C
        mov A,#0FFh                    ;signal
        subb A,sCmdBlock1
        mov cm11,A

        orl CMSEL,#lmoCMSEL1Set        ;Set CM1 to CT
        orl CMEN,#lmoCMEN1Set          ;Set CM1 operational

        ret

;-----
;       Act on SCSI Focus using stepper motor Command
;
;       The lens output is generated focussed near or far.
;-----
;       Command block calling parameters :
;           Byte 1

```

```

;           Byte 2 dir (bit 1 only)
;           Byte 3&4 Number of steps H & L
;
;       SCSI Info sent to Mac :
;-----

r_Focus2:
;       Set up number of steps

        mov R2,sCmdBlock3
        inc R2
        mov R3,sCmdBlock4

;       Check direction

        mov A,sCmdBlock2
        jnb Acc.0,r_Focus2_J1
        m_Focus2Forwards
        jmp r_Focus2_J2

r_Focus2_J1: m_Focus2Reverse
r_Focus2_J2:

;       Complete Focus Steps

r_Focus2_L1: m_Focus2Step_5mS

        djnz R3,r_Focus2_L1
        djnz R2,r_Focus2_L1

        ret

;-----
;       Act on SCSI Focus at infinity using stepper motor Command
;
;       The lens is focussed far until limit switch.
;-----
;       Command block calling parameters :
;       ( Byte 1 ( 0 = Keep control, Else return immediately))
;
;       SCSI Info sent to Mac :
;-----

r_Focus2I:

;       Drive mechanism back until first sensor input is reached

        m_Focus2Reverse

r_Focus2I_L3: m_Focus2Step_5mS

        mov A,P5
        jnb lbaInfinity,r_Focus2I_L3      ; If low, not home yet

        ret

;-----
;       Act on SCSI Reset Device Command
;
;-----
;       Command block calling parameters :
;
;       SCSI Info sent to Mac :
;-----

r_ResetDev:

;       Return Status, wait for an interrupt and check it is a function
;       complete interrupt.

        m_Conclude 00h
        m_WaitSCSIInt
        m_IsIntFC

        ljmp Init                        ;Reset device from scratch

;-----
;       Act on SCSI Power on Unit
;
;-----
;       Command block calling parameters :
;       Byte 1 i=on 0=off
;

```

```

;      SCSI Info sent to Mac :
;
;-----
r_Power:    mov A,sCmdBlock1
            jb Acc.0,r_Power_J1

            setb opPowerSav
            lcall r_SetCamOff
            ret

r_Power_J1: lcall r_SetCamOn
            clr opPowerSav
            ret

;-----
;      Routine to check and set up integration time based on CmdBlk3 & 4
;
;-----
r_SetInteg:
;      Check whether integ time has changed

            mov A,csIntegTimeH
            cjne A,cqIntTimeH,r_SetInt_Set

            mov A,csIntegTimeL
            cjne A,cqIntTimeL,r_SetInt_Set

            ret

;      Do integration time / Stepper Pulse 2 Initialisation
;      Check whether integration time>1024 then a:=10;
r_SetInt_Set:
            mov A,cqIntTimeH
            cjne A,#cxOptIntTimeH,$+08H
            mov A,cqIntTimeL
            cjne A,#cxOptIntTimeL,$+03H
            jc $+5
            ljmp r_SetInt_J1

;      Integration time < 10 mS (<468)
;      - Calculate set time for integration time

            m_Subtr #0FFh, #0FFh, cqIntTimeH, cqIntTimeL
            mov CCH2,orRetByteH ;Set integration time
            mov CCL2,orRetByteL

;      - Calculate Second step value from min constant integration time
            m_Subtr #0FFh, #0FFh, #cxOptIntTimeH, #cxOptIntTimeL
            m_Subtr orRetByteH, orRetByteL, #cxStepDelayH, #cxStepDelayL

            mov CCH1,orRetByteH ;Set Delay B4 2nd step @ 200 pulses
            mov CCL1,orRetByteL

;      - Calculate reload value from integration time
            m_Subtr orRetByteH, orRetByteL, #cxStepDelayH, #cxStepDelayL

            mov TH2,orRetByteH ;Set initial T2 value
            mov TL2,orRetByteL

            mov CRCH,orRetByteH ;Set T2 reload
            mov CRCL,orRetByteL

;      Wait for line synchronisation to settle camera output

            clr TF2
            m_WaitTF2 ;Wait for start of line

            ret

r_SetInt_J1:
;      Integration time > 10 mS (>468)
;      - Calculate set time for integration time

            m_Subtr #0FFh, #0FFh, cqIntTimeH, cqIntTimeL
            mov CCH2,orRetByteH ;Set integration time
            mov CCL2,orRetByteL

;      - Calculate Second step value from integration time
            m_Subtr orRetByteH, orRetByteL, #cxStepDelayH, #cxStepDelayL

            mov CCH1,orRetByteH ;Set Delay B4 2nd step @ 200 pulses
            mov CCL1,orRetByteL

```

```

;      - Calculate reload value from integration time
      m_Subtr orRetByteH, orRetByteL, #cxStepDelayH, #cxStepDelayL

      mov TH2,orRetByteH          ;Set initial T2 value
      mov TL2,orRetByteL

      mov CRCH,orRetByteH        ;Set T2 reload
      mov CRCL,orRetByteL

;      Wait for line synchronisation to settle camera output

      clr TF2
      m_WaitTF2                  ;Wait for start of line

      ret

```

```

;-----
;      Act on SCSI Set A/D Table Command
;-----
;      Command block calling parameters :
;      Byte 1 AD Table to change use (1, 2 or 3)
;-----
;      SCSI Info sent to Mac :
;-----

```

```

r_SetADTable:
      mov A,sCmdBlock1
      cjne A,#3,r_SetAD_J1

      mov csTableAddrH,#HIGH(oTable3_1Base) ;Set up ADTable3
      ret

r_SetAD_J1:
      cjne A,#2,r_SetAD_J2

      mov csTableAddrH,#HIGH(oTable2_1Base) ;Set up ADTable2
      ret

r_SetAD_J2:
      mov csTableAddrH,#HIGH(oTable1_1Base) ;Set up ADTable1
      ret

```

M.2 TreeScan Plug-in Code (Version 3.28)

The TreeScan plug-in is written in C and is divided into seven source files with associated header files and #include libraries.

TreeScan plug-in source files

Main.c	Main TreeScan program file which contains the dispatching routine calling routines in Operations.c dependant on the value of the selector parameter passed from the calling application.
Modification History.c	A comprehensive modification history within a comment.
Operations.c	Contains the main routines for the plug-in. This includes the implementation of the main TreeScan dialog window, as well as code to initialise and complete a plug-in call.
Utilities.c	Library of utility routines.
Debug.c	Implementation of the debug / development dialog window which allows individual commands to be sent to the scanner and tested.
Functions.c	Miscellaneous functions for the TreeScan plug-in. These include all routines to send SCSI commands to the scanner and receive image data.
Integ/Focus.c	Implementation of integration time adjustment routines, autofocus routines, and blind refocus routines.

This appendix contains a complete listing of Main.c, Modification History.c, and Operations.c. Relevant sections have been included from Functions.c and Integ/Focus.c.

M.2.1 Modification History.c Source Listing

/* Modification History

We need to keep a modification history so I'm putting it in a separate file.c so I can make it part of the project. Contains one large comment and no code.

Vers.	Author	Date	Comments
991	MW	27/1/95	Software For Treescan 2 System. Software includes image capture plugin including image capture algorithm, sending of SCSI commands, etc. ie. Fully operational image capture software with modifications for stepper motor control on autofocus & refocus
2.0			New naming convention
2.1	MW	1/2/95	Autofocus algorithm redesigned and reprogrammed
2.2	RHP	1/2/95	Tidied up several return 0's to beep break so dialog box is still active; check viewname file doesn't already exist; Ask user if view 1 or view 2 and if view 2 is is cw or ccw from v1; Ask user for dip and lean angles and save to file treename_v1.dat or treename_v2.dat; send a go home with immediate return when we exit; I have assumed that we have enclosing:something:image folder:image app; and we will store all data and image files in a folder enclosing:Treescan Images; On full scan we check filename does not already exist.; Move down about 70cm before starting scan. So we focus at breast height but capture from 70cm up.
2.3	RHP		vref number is set on fullscan so it gets passed back to image so image will go in the correct folder
2.4	RHP		Refocus options no refocus, autorefocus, blind refocus; displays fnum and fstepno at end of autofocus; loads ftable from file
2.8	...		fixed blindrefocus
2.9			fixed saturation problem by halving int time if new int > 10X old int
3.0			Blind refocus every 250 steps AutoRefocus every 1000 steps Select viewname on entry to plugin dialog turn off refocus data files print wn to log file = 0 unless doing blindrefocus
3.1			wn still was not being printed in log file - fixed partial shots were returning full image - fixed setup image name and vrefnum for preview eheight was > chunksize so we got oscillation as we were looking at stuff before previous correction
3.2			fixed what int routine looks at p-lump to p-lump+eheight
3.3			Set int time again after we move down for start of shot Modify required average light level to be 100 Dont clear focussed at end of preview try and speed up initial set int time by rough guess Modified getline to allow comments in file !lkjlk and dist, fstepnum on each line, dist is ignored
3.4			Try and unload the other segments of code resource on return from finish call
3.5	MW, AD		Conservatism factor of 80% if change > 20%, write aperture data to log files; fixed double integration adjustment bug.
3.6	MW, AD		NoRefocus changes to NoFocus, No adjust integration time button added (each call to Refocus modified), Make preview move back StepsBack lines
3.6b	MW		No aperture adjustment and no focus adjustment during lines back! Fixed the return partial image if capture is cancelled.
3.7	MW		Combined Mk1 / M2 software compatibility.
3.8	RHP		Write variety of focus numbers to file ie 2,4,8 pixel separations; fixed file name for focus data on use of focus button. - not initialised; shortened delays in focus and preview routines; preview is now centered vertically on focus line
3.9	AD		New control for steps perl line in getlines getchunk passes micro # steps to move for each line calls different scsi cmd for getchunk - requires eeprom version 3.9, board revision 3.9
3.10m1	MW		Redevelop plugin with new user friendly user interface. Completed entering new dialog boxes
3.10m2			Aim to get compiling
3.10m3			Video debug dialog operational
3.10m4			Start on debug box
3.10m5			Debug box commands implemented, about to split functions file
3.10m7			Front window commands implemented, correct steps /angle in preview, implemented the hiding of chunklines
3.10m10			View button operating correctly, can use up arrow & down arrow to tab, use enter & return to operate buttons
3.11	MW		Plugin with new interface / structure fully operational
3.11b	MW		In debug aper / focus commands don't turn power off if set on, clarification of focus step buttons, switched display of focus

numbers for autofocus, max distance 1800 steps, set aperture done
 flag in preview & capture.
 3.11d RHP Only store globals after start or finish call
 Allocate storage for globals in heap when we store them not at
 the begining; dont call store globals after we have done
 unloadsega4; reorder allocation of space in start routine so we
 allocate the big bit first.
 3.11e MW fixed 1st char bug, slider getting updated on entry.
 3.11f RHP To work with NIH Image TF 3.3a which tells us about all the free
 mem on entry not only half of it.
 Switch to tenbit automatically if inttime>5000
 only allow the lines we have space for in terms of contiguous
 free mem when we return to NIH
 3.12 RHP Autofocus and blind refocus should be at x/stepsperline to make
 sizes stay same
 3.12b MW fixed overflow in reading number of lines
 3.12c MW Fixed blind focus crashing problem (increased size of temp
 string s)
 3.12e MW fixed bug in log files
 3.12f RHP print better error message on fail in getchunk.
 hold off on adj exposure and refocus should depend on
 stepsperline
 3.16 RHP Reads bat voltage ok . checks before capture
 3.17 RHP Make sure we move forward before going home for focus to insure
 we dont try and drive past the infinity position
 In autofocus dont search back past 0 fstep
 Refocus routine had never been tidied up and bugs fixed - done
 but needs testing still
 3.18 RHP autofocus goes to 10bit if first loop is taking it past 3000
 refocus now prints distance estimate and then fstep
 blind focus was ignoring stepsperline - fixed
 3.19 MW Fixed .dat file problem
 3.20 RHP Was not reading last element of focus table - fixed
 3.20a AD, MW loosing character in dialog box fixed
 3.21 RHP switch back an forward between 8 and 10 bit as we go up tree
 log max min av of pixels in focus log file
 3.22 RHP focusnum - float and use av intensity in calculating it
 ...
 3.25 RHP add calls for changing video amp gain
 3.26 RHP skip reset,powerson etc if control key is down on plugin entry
 3.27 RHP Make it if option key is pressed
 3.28 RHP Auto set video gain to ideal value
 TO DO

Debug code

```

{ Str255 s;
  sprintf((char*)s,"%d,%d,%d Alpha = %f, Dist =
           %d",i,fTable[i],fstepnum,alpha,Dist);
  CtoPstr((char *)s);
  ParamText(s,"\p", "\p ", "\p");
  Alert(MsgAlertID, nil);
}
*/

```

M.2.2 Main.c Source Listing

```

/*****
*****
*   TreeScan Photoshop Plugin Module
*
*   Copyright 1994   Massey University
*
*   Ralph Pugmire
*
*****
*****/

/*****
*
*   File      Main.c
*
*   Contains the main dispatching routine.
*
*****/

#include <SetUpA4.h>
#include <SCSI.h>
#include "AcquireInterface.h"

#include "TreeScan.h"

TGlobals      myGlobals;      //structure for globals that are kept in rsrc
Tpref         pref;           //structure for plugin preference variables
Ptr           base;           //pointer to plugin image memory area

Cursor        WatchCrsr;      /* allocate a 68-byte struct */
CursHandle    cursH;

int           tabItem;         // Current Tabbed item
int           rfnum;           // nth refocus up tree
int           fstepnum;        // current number of fsteps back from infinity
int           vgainstep;       // Current video gain step
double        focusnum, focusnum2, focusnum4, focusnum8;
int           bfsteps, lfsteps, min, max;
float         av;

pascal void main (int selector, AcquireRecordPtr myRecPtr, long *dataPtr, int
                  *resultPtr){
    RememberA0();
    SetUpA4();

    if (FirstTime (dataPtr))
        InitGlobals (dataPtr);
    else
        RestoreGlobals (dataPtr);

    switch (selector)
    {
        case acquireSelectorAbout:
            *resultPtr = DoAbout ();
            break;

        case acquireSelectorPrepare:
            *resultPtr = DoPrepare (myRecPtr);
            break;

        case acquireSelectorStart:
            *resultPtr = DoStart (myRecPtr, dataPtr);
            StoreGlobals (dataPtr);
            break;

        case acquireSelectorContinue:
            *resultPtr = DoContinue (myRecPtr, dataPtr);
            break;

        case acquireSelectorFinish:
            *resultPtr = DoFinish (dataPtr);
            StoreGlobals (dataPtr);
            UnloadA4Seg(0L);
            break;

        default:
            *resultPtr = acquireBadParameters;
    }

    RestoreA4();
}

```

M.2.3 Operations.c Source Listing

```

/*****
 *
 *   File      Operations.c
 *
 *   Contains the main routines for the Treescan plugin
 *
 *****/

#include <Events.h>
#include <Quickdraw.h>
#include <Files.h>
#include <Script.h>
#include <ToolUtils.h>
#include <stdio.h>
#include <string.h>
#include <SCSI.h>
#include <Memory.h>
#include "Utilities.h"
#include "AcquireInterface.h"

#include "TreeScan.h"
#include "math.h"

extern TGlobals      myGlobals;
extern Tpref         pref;
extern Ptr           base;

extern Cursor        WatchCrsr;           // allocate a 68-byte struct
extern CursorHandle  cursH;

extern int            tabItem;
extern int            rfnum;              // nth refocus up tree
extern int            fstepnum;          // current no of fsteps back from infinity
extern float          focusnum, focusnum2, focusnum4, focusnum8;

// Operations file globals
int      gotlines;
int      gotImage;           // 1=> We have an image ready for transfer to NIH
int      GNextRow;          // NextRow to send to Image
char      linebuf[1100];    // space for one line from camera
int      partialheight, capturedheight;

// Displays the about box for the module
// -----
int DoAbout (void){
    short      item;
    DialogPtr   myDialogPtr;
    DialogTHndl myDialogTHndl;

    myDialogTHndl = (DialogTHndl) GetResource ('DLOG', AboutDialogID);
    HNoPurge ((Handle) myDialogTHndl);
    CenterDialog (myDialogTHndl);

    myDialogPtr = GetNewDialog (AboutDialogID, nil, (DialogPtr) -1);
    ModalDialog (nil, &item);
    DisposDialog (myDialogPtr);

    HPurge ((Handle) myDialogTHndl);

    return noErr;
}

// Reduces the memory set aside for the module if possible
// -----
int DoPrepare (AcquireRecordPtr myRecPtr){
    long maxmem;

    maxmem = myRecPtr->maxData;
    maxmem = (maxmem/2);
    if (maxmem>(((long)lwidth * Init_Lines)+50000)){
        maxmem = (((long)lwidth * Init_Lines)+50000);
        pref.maxlines = Init_Lines;
    }else
        pref.maxlines = ((maxmem - 50000 )/ lwidth);
    myRecPtr->maxData = maxmem;
    return noErr;
}

// Determines image parameters and informs the host
// -----

```

```

int DoStart (AcquireRecordPtr myRecPtr, long *dataPtr){

    Tstat          scannerstatus;          // Scanner status structure
    sstatPtr       sstat;

    DialogTHndl    myDialogTHndl;          // \
    DialogPtr      myDialogPtr;            // | Control of dialog box
    short          item;                    // |
    int            done;                    // |
    EventRecord    theEvent;                // Cancel image capture event

    ControlHandle  StepAdjScroll,toggleButton; // \
    short          toggleType, StepAdjType; // |
    Rect           toggleBox, StepAdjBox;     // /

    int            err;                      // \
    char           buffer[512];              // | Vars for SCSI commands
    SCSIInstr      myTIB[12];               // |
    cmdblk         mycmd;                    // |
    OSErr          errors[6];                // |
    short          stat, message;            // /

    char           s[256];                   // Temporary string
    long           templong;

    int            i, viewPoint,lastItem;
    long           count;
    float          dip,lean,tilt,bat_volts;
    Str255         lfname;
    FSSpec         infile, outfile;
    short          inrefNum, outrefNum;
    EvQEl          *myPtr;
    StatusBlock    sb;

    if ((base = NewPtr((long)lwidth * pref.maxlines)) == 0 ) return memFullErr;
    if (MaxBlock() < ((long)lwidth * pref.maxlines + 50000) ){
        SysBeep(1);
        pref.maxlines = (MaxBlock()-50000)/lwidth;
    }

    cursH = GetCursor( watchCursor );        /* constant in ToolboxUtil.h */
    HLock ((Handle) cursH);
    WatchCrsr = **cursH;                     /* copy the data */
    HUnlock ((Handle) cursH);

    sstat = &scannerstatus;
    partialheight = 0;
    sstat->intsteps = myGlobals.intsteps;
    if (myGlobals.height>pref.maxlines){
        myGlobals.height = pref.maxlines;
    }

    SetupStatus();                          // Setup plugin preferences

    myDialogTHndl = (DialogTHndl) GetResource ('DLOG', VideoDialogID);
    HNoPurge ((Handle) myDialogTHndl);
    myDialogPtr = GetNewDialog (VideoDialogID, nil, (DialogPtr) -1);

    CenterDialog (myDialogTHndl);
    if (pref.Devt)
        SizeWindow(myDialogPtr,580,420,true);
    else
        SizeWindow(myDialogPtr,580,375,true);
    SetPort(myDialogPtr);

    EstimateLines(myDialogPtr);

    SetDString(myDialogPtr, VersionID, SWVersion);
    SetDNum(myDialogPtr, TreeHeightID, myGlobals.TreeHeight);
    SetDNum(myDialogPtr, StepAdjtxtID, myGlobals.StepAdj);

    GetDItem (myDialogPtr, StepAdjID, &StepAdjType, (Handle*)&StepAdjScroll,
              &StepAdjBox);
    SetCtlValue (StepAdjScroll, myGlobals.StepAdj);

    SelIText (myDialogPtr, ViewNameID, 0, 32767);

    if (!isPressed(OptionKey)){ // if control key is down skip startup calls
        DoReset();              // Send reset to micro
        DoPowerOn();
        Delay(30,&templong);

        // Check supply level before we go
        GetStatus(&sb);
        bat_volts = sb.s.voltage / 1023.0 * v_calib;
        if (bat_volts<11){
            Str255 s;

```

```

        SysBeep (1);
        DoPowerOff();
        sprintf((char*)s, "Battery Voltage is two low = %.1f", bat_volts);
        CtoPstr((char *)s);
        ParamText(s, "\p", "\p ", "\p");
        Alert(MsgAlertID, nil);
        GotImage=0; done = 0;
        DisposPtr(base);
        DisposDialog (myDialogPtr);
        HPurge ((Handle) myDialogTHndl);
        return 1;
    }
    sstat->AtHome = DoHomeRetPwr();
}

sstat->viewnum = 1; viewPoint = v1;
SetViewButton( myDialogPtr, viewPoint, sstat);

sstat->Focused = sstat->IntegOK = sstat->AtHome = sstat->CameraOn = tabItem = 0;
GotImage = done = false;
lastItem = ViewNameID;

do{
    InitCursor();
    ModalDialog ((ModalFilterProcPtr) MyEventFilter, &item);

    for(i=0; i<(max_tabitems-1); i++)
        if ((item==tabItems[i]) && (item!=lastItem))
            tabItem=SetTab(myDialogTHndl, myDialogPtr, i, tabItem, false);
    lastItem = item;

    switch (item){
        case returnID:
            done = true;
            break;

        case DevtID:
            pref.Devt = !pref.Devt;
            if (pref.Devt)
                SizeWindow(myDialogPtr, 580, 420, true);
            else
                SizeWindow(myDialogPtr, 580, 375, true);
            break;

        case TabID:
            tabItem=SetTab(myDialogTHndl, myDialogPtr, tabItem+1, tabItem,
                           false);
            lastItem = tabItems[tabItem];
            break;

        case backTabID:
            tabItem=SetTab(myDialogTHndl, myDialogPtr, tabItem-1, tabItem,
                           false);
            lastItem = tabItems[tabItem];
            break;

        case cmdTabID:
            tabItem=SetTab(myDialogTHndl, myDialogPtr, tabItem+1, tabItem,
                           true);
            lastItem = tabItems[tabItem];
            break;

        case cmdbackTabID:
            tabItem=SetTab(myDialogTHndl, myDialogPtr, tabItem-1, tabItem,
                           true);
            lastItem = tabItems[tabItem];
            break;

        case ViewNumID2:
        case ViewIconID:
        case v1ID:
        case v2cwID:
        case v2ccwID:

            if (++viewPoint > 3) viewPoint = 1;
            SetViewButton( myDialogPtr, viewPoint, sstat);
            break;

        case TreeHeightID:
            {
                int tHeight;
                tHeight = GetDNum(myDialogPtr, TreeHeightID);
                if (tHeight<0 || tHeight>maxtHeight){
                    if (tHeight>maxtHeight)
                        tHeight=maxtHeight;
                    else

```

```

        tHeight=0;
        SetDNum(myDialogPtr, TreeHeightID, tHeight);
        SelIText (myDialogPtr, TreeHeightID, 0, 32767);
        SysBeep(1);
    }
    myGlobals.TreeHeight = tHeight;
    EstimateLines(myDialogPtr);

    break;
}

case TreeHeightUpID:
{
    int tHeight;
    tHeight = myGlobals.TreeHeight + HeightStep;
    SelIText (myDialogPtr, TreeHeightID, 0, 0);
    if (tHeight>maxtHeight){
        tHeight=maxtHeight;
        SelIText (myDialogPtr, TreeHeightID, 0, 32767);
        SysBeep(1);
    }
    SetDNum(myDialogPtr, TreeHeightID, tHeight);
    myGlobals.TreeHeight = tHeight;
    EstimateLines(myDialogPtr);
}
break;

case TreeHeightDwnID:
{
    int tHeight;
    tHeight = myGlobals.TreeHeight - HeightStep;
    SelIText (myDialogPtr, TreeHeightID, 0, 0);
    if (tHeight<0){
        tHeight=0;
        SelIText (myDialogPtr, TreeHeightID, 0, 32767);
        SysBeep(1);
    }
    SetDNum(myDialogPtr, TreeHeightID, tHeight);
    myGlobals.TreeHeight = tHeight;
    EstimateLines(myDialogPtr);
}
break;

case StepAdjID:
    SetDNum (myDialogPtr, StepAdjtxtID, myGlobals.StepAdj);
    EstimateLines(myDialogPtr);
    break;

case PreviewID:{
    int lump,p;

    DoPowerOn();
    SetCursor(&WatchCrsr);
    GotImage = 0;

    GetDString(myDialogPtr, ViewNameID, sstat->ViewName);
    strncpy(&s[1],(char*)&sstat->ViewName[1],sstat->ViewName[0]);
    s[0] = sstat->ViewName[0];
    PtoCstr((unsigned char *)s);
    if (sstat->viewnum==1)
        strcat(s,"_v1");
    else
        strcat(s,"_v2");
    strcpy((char *)myRecPtr->filename,s);
    CtoPstr((char *)myRecPtr->filename);
    sprintf((char*)lfname,":::Treescan Images:%s",s);
    CtoPstr((char*)lfname);
    if (!FSMakeFSSpec(0,0,lfname,&outfile)){
        // File exists so beep and ask do you want to overwrite
        ParamText("\pTree Image ",myRecPtr->filename, "\p already
        exists.",
        "\p\rPlease use another name");
        Alert(MsgAlertID, nil);
        tabItem = SetTab(myDialogTHndl, myDialogPtr, 0, tabItem, false);
        SysBeep(1);
        break;
    }
    myRecPtr->vRefNum = outfile.vRefNum;

    // Move up to focus centre halve preview lines above home
    DoSteps((previewlines/2);

    if (!sstat->IntegOK && (pref.IntegAdj>=NoAdjInteg) ){
        SetDString(myDialogPtr, StatusID, s integ);
        sstat->intsteps = DoAdjInt (myDialogPtr, sstat->intsteps);
        sstat->IntegOK = true;
    }
}

```



```

if (pref.IntegAdj==FixedInteg ){
    sstat->intsteps = pref.FixedIntegTime;
    sprintf((char*)s,"%d ", sstat->intsteps);
    SetDString(myDialogPtr, StatIntNoID, CtoPstr(s));
}
if (!sstat->Focused && (pref.focus>=NoRefocus) ){
    SetDString(myDialogPtr, StatusID, s_focus);
    sstat->Focused = DoFocus(myDialogPtr, sstat);
}

// Move down a bit to centre focus point
Delay(3,&templong);
DoSteps(-(previewlines/2+20));
Delay(3,&templong);
DoSteps(20);

SetDString(myDialogPtr, StatusID, s_capt);
for (p=(previewlines/myGlobals.StepAdj); p>0; p -= chunklines){
    if (WaitNextEvent(mouseDown|keyDown, &theEvent, 1, nil) )
        break;
    if (p<chunklines)
        lump = p;
    else
        lump = chunklines;
    if (GetChunk(lump, p-lump, sstat->intsteps, myGlobals.StepAdj)<4){
        // scsi error
        SysBeep(1);
        break;
    }
    Display(p-lump,p,(previewlines/myGlobals.StepAdj),0,pref.detail,
        myDialogPtr);
}
sstat->AtHome = false;
Delay(3,&templong);
SetDString(myDialogPtr, StatusID, s_home);
DoHome();
DoPowerOff();
SetDString(myDialogPtr, StatusID, s_idle);
GotImage = 1;
// done=1; // Set to capture preview image
FlushEvents(CxFFFF, 0);
break;
}

case FullScanID:{
    int af,ef,p,lump,wn;
    Byte *pp;
    long startticks,ticks;
    char s[256];
    int dfirst,dlast,dinc,i,d,Dist,ftable[500];
    float o,alpha;
    int height;

    // check dip is valid in case we need it for blind refocus
    dip = GetDReal(myDialogPtr,DipID);
    if (dip>90 || dip<-90){
        ParamText("\pDip angle invalid. ", "\p", "\p ",
            "\p\rPlease re enter");
        Alert(MsgAlertID, nil);
        SysBeep(1);
        break;
    }
    tilt = GetDReal(myDialogPtr,TiltID);
    if (tilt>90 || tilt<-90){
        ParamText("\pTilt angle invalid. ", "\p", "\p ",
            "\p\rPlease re enter");
        Alert(MsgAlertID, nil);
        SysBeep(1);
        break;
    }
    SetCursor(&WatchCrsr);
    rfnun = 0;
    done = 1; // Return at end of image unless reset in the loop

// ----- Check image file exists & Setup log file -----
    GetDString(myDialogPtr, ViewNameID, sstat->ViewName);
    strncpy(s[1],(char*)&sstat->ViewName[1],sstat->ViewName[0]); s[0] =
        sstat->ViewName[0];
    PtoCstr((unsigned char *)s);
    if (sstat->viewnum==1)
        strcat(s,"_v1");
    else
        strcat(s,"_v2");
    strcpy((char *)myRecPtr->filename,s);
    CtoPstr((char *)myRecPtr->filename);
    sprintf((char*)lfname,":::Treescan Images:%s",s);
    CtoPstr((char*)lfname);
}

```

```

if (!FSMakeFSSpec(0,0,lfname,&outfile)){
    // File exists so beep and ask do you want to overwrite
    ParamText("\pTree Image ",myRecPtr->filename, "\p already
        exists.",
        "\p\rPlease use another name");
    Alert(MsgAlertID, nil);
    tabItem = SetTab(myDialogTHndl, myDialogPtr, 0, tabItem, false);
    SysBeep(2); done = 0; break;
}
myRecPtr->vRefNum = outfile.vRefNum;
if (pref.LogFiles){
    sprintf((char*)lfname,":::Treescan Images:%s.log",s);
    CtoPstr((char*)lfname);
    err = FSMakeFSSpec(0,0,lfname,&outfile);
    err = FSpDelete(&outfile);
    err = FSpCreate(&outfile,'Imag','TEXT',smSystemScript);
    if (err) {SysBeep(2); done = 0; break;}
    err = FSpOpenDF(&outfile,fsCurPerm,&outrefNum);
    if (err) {SysBeep(2); done = 0; break;}
}

// -----

if (pref.focus == BlindRefocus)
    af = bafchunks/myGlobals.StepAdj;
else
    af = afchunks/myGlobals.StepAdj;
Dist = 0;

DoPowerOn();
Delay(120,&templong);
// Check supply level before we go
GetStatus(&sb);
bat volts = sb.s.voltage / 1023.0 * v_calib;
if (bat_volts<11){
    Str255 s;

    SysBeep (1);
    DoPowerOff();
    sprintf((char*)s,"Battery Voltage is two low = %.1f",bat_volts);
    CtoPstr((char *)s);
    ParamText(s,"\p", "\p ", "\p");
    Alert(MsgAlertID, nil);
    GotImage=0; done = 0; break;
}

// Move up to focus centre halve preview lines above home
DoSteps(previewlines/2);

// ----- Set integration & perform autofocus -----
if (!sstat->IntegOK && (pref.IntegAdj>=NoAdjInteg) ){
    SetDString(myDialogPtr, StatusID, s_integ);
    sstat->intsteps = DoAdjInt(myDialogPtr,sstat->intsteps);
    sstat->IntegOK = true;
}
if (pref.IntegAdj==FixedInteg ){
    sstat->intsteps = pref.FixedIntegTime;
    sprintf((char*)s,"%d ", sstat->intsteps);
    SetDString(myDialogPtr, StatIntNoID, CtoPstr(s));
}
if (!sstat->Focused && (pref.focus>=NoRefocus) ){
    SetDString(myDialogPtr, StatusID, s_focus);
    sstat->Focused = DoFocus(myDialogPtr, sstat);
}

// ----- Start setup for blind refocus -----
if (true) { // Always read blindfocus table
    // First load the ftable from file Treescan.ftable
    err = FSMakeFSSpec(0,0,"\pTreescan.ftable",&infile);
    err = FSpOpenDF(&infile,fsCurPerm,&inrefNum);
    if (err) {
        ParamText("\pCant open blind refocus data file","\p", "\p ",
            "\p");
        Alert(MsgAlertID, nil);
        SysBeep(1); done = 0; break;
    }

    //get next line from the file
    if (getline(inrefNum,s,100)!=noErr){
        ParamText("\pError reading ftable file","\p", "\p ", "\p");
        Alert(MsgAlertID, nil);
        SysBeep(1); done = 0; break;
    }
    sscanf(s, "%f,%d,%d,%d", &alpha, &dfirst, &dlast, &dinc);

    //Make space for the table

    for (i=0; i<=((dlast-dfirst)/dinc); i++){

```

```

// get next line from the file
if (getline(inrefNum,s,100)!=noErr){
    ParamText("\pError reading ftable file","\p", "\p ", "\p");
    Alert(MsgAlertID, nil);
    SysBeep(1); done = 0; break;
}
sscanf(s, "%d,%d", &d, &ftable[i]);
if ((dfirst+i*dinc)!=d){
    Str255 s;
    sprintf((char*)s,"Error in focus table line %d
    %d<>%d",i,d,dfirst+i*dinc);
    CtoPstr((char *)s);
    ParamText(s,"\p", "\p ", "\p");
    Alert(MsgAlertID, nil);
    break;
}
}
FSClose(inrefNum);

// find our distance
for (i=0; i<((dlast-dfirst)/dinc); i++)
    if (ftable[i] <= fstepnum) break;
Dist = dfirst + (i * dinc);

if (Dist>2500) Dist = 1500;
sprintf((char*)s,"Dist: %3.2f m", (float)Dist/100);
SetDString(myDialogPtr, StatFocusNo3ID, CtoPstr(s));

//calc lines;
myGlobals.height = HeighttoLines(myGlobals.TreeHeight, dip, Dist);
sprintf((char*)s,"%d (calc)", myGlobals.height);
SetDString(myDialogPtr, StatLineNoID, CtoPstr(s));

if (myGlobals.height>pref.maxlines){
    Str255 s;
    myGlobals.height = pref.maxlines;
    myGlobals.TreeHeight =
        LinestoHeight(myGlobals.height,dip,Dist)/100;
    sprintf((char*)s,"Can only capture to %4.1f metres high.",
        LinestoHeight(myGlobals.height,dip,Dist)/100);
    CtoPstr((char *)s);
    ParamText(s,"\p", "\p ", "\p");
    Alert(MsgAlertID, nil);
    SetDNum(myDialogPtr, TreeHeightID, myGlobals.TreeHeight);
}
}
height = myGlobals.height;

// ----- End setup for blind refocus -----

GetDItem (myDialogPtr, FullScanID, &toggleType,
    (Handle*)&toggleButton, &toggleBox);
SetCTitle (toggleButton, "\pReturn Image");
GetDItem (myDialogPtr, CancelID, &toggleType,
    (Handle*)&toggleButton, &toggleBox);
SetCTitle (toggleButton, "\pAbort");

ef = echunks;
GotImage = 2;
startticks = TickCount();

// Move down a bit to start picture about 0.7 below focus point
Delay(30,&templong);
DoSteps(-(StepsBack+20));
Delay(30,&templong);
DoSteps(20);

// ***** Start image capture loop *****
for (p=height; p>0; p -= chunklines){
    SetDString(myDialogPtr, StatusID, s_capt);

    sprintf((char*)s,"%d of %d ",(height-p), myGlobals.height);
    SetDString(myDialogPtr, StatLineNoID, CtoPstr(s));

    sprintf((char*)s,"Capturing image at %4.1f m
    ",LinestoHeight((height-p),dip,Dist)/100);
    SetDString(myDialogPtr, StatusID, CtoPstr(s));

// ***** Check for user image capture break *****
if (WaitNextEvent(mouseDown|keyDown, &theEvent, 1, nil) ) {
    // Check if the event received is Stop or else cancel
    Rect itemBox;
    short itemType;
    Point thePoint;
    ControlHandle theControl, itemHandle;

```

```

        GetDItem (myDialogPtr, FullScanID, &itemType,
                  (Handle*)&itemHandle, &itemBox);
        thePoint = theEvent.where;
        GlobalToLocal (&thePoint);
        FindControl (thePoint, myDialogPtr, &theControl);
        if (theControl == itemHandle && p<height){ //Stop
            gotlines = height-p;
            GotImage = 3;
            break;
        }
        else{ //Cancel
            gotlines = height-p;
            GotImage = 3;
            done = 0;
            break;
        }
    }

// ***** Capture one chunk *****
    if (p<chunklines)
        lump = p;
    else
        lump = chunklines;
    if (GetChunk(lump, p-lump, sstat->intsteps, myGlobals.StepAdj)<4){
        // scsi error
        SysBeep (1);
        sprintf((char*)s, "Error in Get Chunk");
        ParamText (CtoPstr(s), "\p", "\p ", "\p");
        GotImage=0; done = 0; break;
    }
    if (pref.ChunkMarks)
        for (pp = (Byte*)((long)p*lwidth)+base+250;
             pp<(Byte*)((long)p*lwidth)+base+300; pp++)
            *pp = 255;
    Display(p-lump, p, height, 0, pref.detail, myDialogPtr);

// ***** Store relevant information to log file *****
    if (pref.LogFiles){
        ticks = TickCount();
        o = ((height-p+lump/2+chunklines)*alpha*myGlobals.StepAdj) +
            dip;
        wn = Dist/cos(o/180*3.14159);

        sprintf((char*)s, "\r%d %d %d %d %d %d %.1f %d %d %d ",
                p, ticks-startticks, af, ef, sstat->intsteps, focusnum, rfnum, fstepnum, wn);
        CtoPstr(s); count = s[0];
        err = FSWrite(outrefNum, &count, &s[1]);
        if (err) {SysBeep(2); FSClose(outrefNum); done = 0; break;}
    }
    if ((p-chunklines) <= 0) break;

// ***** Adjust integration and do refocus *****
    if (!(--af)){
        rfnum++;
        ef = echunks;
        if ((pref.IntegAdj == AdjInteg) && ((height-p)>(StepsBack/myGlobals.StepAdj)) )
            sstat->intsteps = DoAdjInt2(myDialogPtr, sstat->intsteps, p-lump, outrefNum);
        if (pref.focus == AutoRefocus){
            af = afchunks/myGlobals.StepAdj;
            if ((height-p)>(StepsBack/myGlobals.StepAdj)){
                SetDString(myDialogPtr, StatusID, s_adjfocus);
                sstat->Focused = DoReFocus(myDialogPtr, p+lump, sstat);
            }
        }
        if (pref.focus == BlindRefocus){
            af = bafchunks/myGlobals.StepAdj;
            if ((height-p)>(StepsBack/myGlobals.StepAdj)){
                SetDString(myDialogPtr, StatusID, s_adjfocus);
                sstat->Focused = DoBlindRefocus(myDialogPtr, p+lump,
                    alpha, Dist, dip,
                    height-p+lump/2+chunklines, ftable, dfirst, dinc,
                    dlast, sstat);
            }
        }
    }
    else if (!(--ef)){
        ef = echunks;
        if ((pref.IntegAdj == AdjInteg) && ((height-p)>(StepsBack/myGlobals.StepAdj)) )
            sstat->intsteps = DoAdjInt2(myDialogPtr, sstat->intsteps, p-lump, outrefNum);
    }

// ***** End of image capture loop *****

```

```

    }
    if (pref.ChunkMarks)
        for (pp = (Byte*)((long)(height-
            StepsBack/myGlobals.StepAdj)*lwidth)+base+lmid -
            fwidth/2 + foffset);
            pp<(Byte*)((long)(height-
            StepsBack/myGlobals.StepAdj)*lwidth)+base+lmid +
            fwidth/2 + foffset); pp++)
        *pp = 255;

    if (pref.LogFiles) FSClose(outrefNum);

    GetDlgItem (myDialogPtr, FullScanID, &toggleType,
        (Handle*)&toggleButton, &toggleBox);
    SetCTitle (toggleButton, "\pCapture Image");
    GetDlgItem (myDialogPtr, CancelID, &toggleType,
        (Handle*)&toggleButton, &toggleBox);
    SetCTitle (toggleButton, "\pCancel");

    FlushEvents(0xFFFF, 0);

    if (!done) {
        SetDString(myDialogPtr, StatusID, s_home);
        DoHome();
        DoPowerOff();
        sstat->AtHome = true;
        sstat->Focused = false;
        sstat->IntegOK = false;
    }
    else
        DoHomeRetPwr();
    SetDString(myDialogPtr, StatusID, s_idle);
    break;
}

case CancelID:
    done = 1;
    break;

case HomeID:
    SetCursor(&WatchCrsr);
    DoPowerOn();
    sstat->AtHome = DoHome();
    DoPowerOff();
    break;

case DebugID:
    Debug (sstat);
    EstimateLines(myDialogPtr);
    SetDNum(myDialogPtr, TreeHeightID, myGlobals.TreeHeight);
    break;
}

}while (!done);

DisposDialog (myDialogPtr);
HPurge ((Handle) myDialogTHndl);

capturedheight = myGlobals.height;

myRecPtr->imageMode = 1;
myRecPtr->imageSize.h = myGlobals.width;
myRecPtr->rowBytes = lwidth;
myRecPtr->imageSize.v = capturedheight;
myRecPtr->depth = 8;
myRecPtr->planes = 1;
myRecPtr->data = nil;

if (item != CancelID){
    myGlobals.intsteps = sstat->intsteps;
    myGlobals.height = capturedheight;
    StorePrefs ();
    GNextRow = 0;
    if (GotImage==1){
        myRecPtr->imageSize.v = previewlines/myGlobals.StepAdj;
        capturedheight = previewlines/myGlobals.StepAdj;
    }
    if (GotImage==3){
        myRecPtr->imageSize.v = gotlines;
        partialheight = capturedheight - gotlines;
        capturedheight = gotlines;
        SysBeep(1);
    }
}

if (GotImage){
    // create the .dat file with dip,lean and v2rot in it.
    if (tilt>0)

```

```

        lean = 90-tilt;
    else
        lean = -(90+tilt);
    strncpy(&s[1], (char*)&sstat->ViewName[1], sstat->ViewName[0]);
    s[0] = sstat->ViewName[0];
    PtoCstr((unsigned char *)s);
    if (sstat->viewnum==1)
        strcat(s, "_v1");
    else
        strcat(s, "_v2");
    sprintf((char*)lfname, "::::Treescan Images:%s.dat", s);
    CtoPstr((char*)lfname);
    err = FSMakeFSSpec(0, 0, lfname, &outfile);
    err = FSpDelete(&outfile);
    if (FSpCreate(&outfile, 'Imag', 'TEXT', smSystemScript))
        SysBeep(2);
    if (FSpOpenDF(&outfile, fsCurPerm, &outrefNum))
        SysBeep(2);
    if (sstat->viewnum==1)
        sprintf((char*)s, "%d\\r%f\\r%f\\r", myGlobals.StepAdj, dip, lean);
    else
        sprintf((char*)s, "%d\\r%f\\r%f\\r%d\\r", myGlobals.Step
            Adj, dip, lean, sstat->v2rot);
    CtoPstr(s); count = s[0];
    FSWrite(outrefNum, &count, &s[1]);
    FSClose(outrefNum);

    return noErr;
}

DisposPtr(base); //do it here as finish only called if return without error
return 1;
}

// Returns the data to the host
int DoContinue (AcquireRecordPtr myRecPtr, long *dataPtr){
    long count;

    if (!myGlobals.width)
        return 1;

    if (!GotImage)
        return 1;

    if (CallPascalB (myRecPtr->abortProc))
        return 1;

    CallPascal (GNextRow, capturedheight, myRecPtr->progressProc);

    if (GNextRow >= capturedheight){
        myRecPtr->data = nil;
        SetRect (&myRecPtr->theRect, 0, 0, 0, 0);
        return 0;
    }

    count = (myRecPtr->maxData - (pref.maxlines*lwidth))/ lwidth;
    if (count < 1)
        return memFullErr;

    if (count > capturedheight - GNextRow)
        count = capturedheight - GNextRow;

    SetRect (&myRecPtr->theRect, 0, GNextRow, myGlobals.width, GNextRow+count);

    myRecPtr->loPlane = 0;
    myRecPtr->hiPlane = 0;
    myRecPtr->colBytes = 1;
    myRecPtr->rowBytes = lwidth;
    if ( GotImage == 3 )
        myRecPtr->data = base + ((long)partialheight * lwidth) +
            ((long)GNextRow * lwidth) + lmid - myGlobals.width/2;
    else
        myRecPtr->data = base + ((long)GNextRow * lwidth) + lmid - myGlobals.width/2;

    GNextRow += count;

    return noErr;
}

// Clears the permanent variables
int DoFinish (long *dataPtr){
    *dataPtr = 0;
    DisposPtr(base);
    return noErr;
}

```

M.2.4 Extracts from Functions.c Source Listing

```

/*****
*
*   File      Functions.c
*
*   Contains miscellaneous functions for Treescan plugin
*
*****/

...

//-----
// Do Home mirror mechanism
//-----
int DoHome(){
    int          err;
    cmdblk       mycmd;
    OSErr        errors[6];
    short        stat, message;
    Str255       s;

    mycmd.OpCode = scmResetMech;
    mycmd.r1 = mycmd.r2 = mycmd.r3 = mycmd.r4 = mycmd.r5 = 0;
    err = SCSI_SendCommand_s(&mycmd, myGlobals.SCSI_ID, errors, &stat, &message);
    if (err<3){
        NumToString(err,s);
        ParamText("\pSCSI Error count=",s, "\p in home command", "\p");
        Alert(MsgAlertID, nil);
        return 0;
    }
    return 1;
}

//-----
// Do Home with power off and immediate return
//-----
int DoHomeRetPwr(){
    int          err;
    cmdblk       mycmd;
    OSErr        errors[6];
    short        stat, message;
    Str255       s;

    mycmd.OpCode = scmResetMech;
    mycmd.r1 = mycmd.r2 = 1;
    mycmd.r3 = mycmd.r4 = mycmd.r5 = 0;
    err = SCSI_SendCommand_s(&mycmd, myGlobals.SCSI_ID, errors, &stat, &message);
    if (err<3){
        NumToString(err,s);
        ParamText("\pSCSI Error count=",s, "\p in home command", "\p");
        Alert(MsgAlertID, nil);
        return 0;
    }
    return 1;
}

//-----
// Reset Micro
//-----
void DoReset(){
    int          err;
    cmdblk       mycmd;
    OSErr        errors[6];
    short        stat, message;
    Str255       s;

    mycmd.OpCode = scmResetDev;
    mycmd.r1 = mycmd.r2 = mycmd.r3 = mycmd.r4 = mycmd.r5 = 0;
    err = SCSI_SendCommand(&mycmd, myGlobals.SCSI_ID, errors, &stat, &message);
    if (err<4){
        NumToString(err,s);
        ParamText("\pSCSI Error Count=",s, "\p in reset command", "\p");
        Alert(MsgAlertID, nil);
    }
}

...

```



```

// -----
// Move Focus
// Decide whether the Mk1 or Mk2 focus routine should
// be used.
// -----
void DoMoveFocus(steps)
{
    if ( pref.Mk2 )
        DoMoveFocusMk2(steps);
    else
        DoMoveFocusMk1(steps*1); // x Adjustment factor btwn mk1 & mk2
}

// -----
// Move Focus Mk1
// -----
void DoMoveFocusMk1(steps){
    int          err,dirn;
    cmdblk       mycmd;
    OSErr        errors[6];
    short        stat, message;
    Str255       s;

    if (steps<0){
        steps = -steps;
        dirn = 0;
    }else
        dirn = 1;
    mycmd.OpCode = scmFocus;
    mycmd.r1 = 0;
    mycmd.r2 = dirn;
    mycmd.r3 = steps/256;
    mycmd.r4 = steps%256;
    mycmd.r5 = 0;

    err = SCSIISendCommand(&mycmd, myGlobals.SCSI_ID, errors, &stat, &message);
    if (err<4){
        NumToString(err,s);
        ParamText("\pSCSI Error Count=",s, "\p in Adjust focus command", "\p");
        Alert(MsgAlertID, nil);
    }
}

// -----
// Move Focus Mk2 (Stepper controlled focus)
// -----
void DoMoveFocusMk2(steps){
    int          err,dirn;
    cmdblk       mycmd;
    OSErr        errors[6];
    short        stat, message;
    Str255       s;
    long         x;

    if (steps>1800){SysBeep(1); return;}
    if (steps<-1800){SysBeep(1); return;}
    if (steps==0) return;
    fstepnum += steps;
    if (steps<0){
        steps = -steps;
        dirn = 0;
    }else
        dirn = 1;
    mycmd.OpCode = scmFocus2;
    mycmd.r1 = 0;
    mycmd.r2 = dirn;
    mycmd.r3 = steps/256;
    mycmd.r4 = steps%256;
    mycmd.r5 = 0;

    err = SCSIISendCommand(&mycmd, myGlobals.SCSI_ID, errors, &stat, &message);
    if (err<4){
        NumToString(err,s);
        ParamText("\pSCSI Error Count=",s, "\p in Adjust focus command", "\p");
        Alert(MsgAlertID, nil);
    }
}

// Delay(1,&x); // let it settle
}

...

// -----
// Send a SCSI Command
// returns number of succesful steps
// -----

```

```

int SCSI_SendCommand(cmdblk *mycmd, int SCSI_ID, OSErr *err, short *status, short
    *message){
    int i, rerror;

    for (i=0; i<5; i++) err[i] = 0;
    *status = *message = 0;

    if ((err[0] = SCSI_Get()) != noErr) return 0;
    if ((err[1] = SCSI_Select(SCSI_ID)) != noErr) return 1;
    if ((err[2] = SCSI_Cmd((Ptr)mycmd,6)) == noErr){ rerror = 3; }else rerror = 2;
    if ((err[4] = SCSI_Complete(status,message,TimeOut)) == noErr)
        if (rerror==3) return 4;
    return rerror;
}

//-----
// Send a SCSI Command with read command
// return error is number of sequential succesful steps 4=perfect
//-----
int SCSI_DataCommand(cmdblk *mycmd, SCSI_Instr *myTIB, int SCSI_ID, OSErr *err, short
    *status, short *message){
    int i, rerror;

    for (i=0; i<5; i++) err[i] = 0;
    *status = *message = 0;

    if ((err[0] = SCSI_Get()) != noErr) return 0;
    if ((err[1] = SCSI_Select(SCSI_ID)) != noErr) return 1;
    if ((err[2] = SCSI_Cmd((Ptr)mycmd,6)) == noErr){
        rerror = 3;
        if ((err[3] = SCSI_Read((Ptr)myTIB)) == noErr) rerror = 4;
    }else
        rerror = 2;
    if ((err[4] = SCSI_Complete(status,message,TimeOut)) == noErr)
        if (rerror==4) return 5;
    return rerror;
}

...

// -----
// Get a chunk of x lines and store in buffer with an offset
// -----
int GetChunk(int Lines, long offset, int int_time, int stepsperline){
    int err;
    SCSI_Instr myTIB[15];
    cmdblk mycmd;
    OSErr errors[6];
    short stat, message;
    Ptr bufferPtr;
    long Dummy;

    if (pref.AD10)
        mycmd.OpCode = scmGetXLinesN3;
    else
        mycmd.OpCode = scmGetXLines8bm;
    mycmd.r1 = Lines/256;
    mycmd.r2 = Lines%256;
    mycmd.r3 = int_time/256;
    mycmd.r4 = int_time%256;
    mycmd.r5 = stepsperline;

    bufferPtr = base + (offset*lwidth);
    if (Lines) bufferPtr += (long)(Lines-1) * lwidth;

    myTIB[0].scOpcode = scNoInc;
    myTIB[0].scParam1 = (unsigned long) bufferPtr;
    myTIB[0].scParam2 = 1;

    myTIB[1].scOpcode = scNoInc;
    myTIB[1].scParam1 = (unsigned long) bufferPtr;
    myTIB[1].scParam2 = lwidth;

    myTIB[2].scOpcode = scNoInc;
    myTIB[2].scParam1 = (unsigned long) &myTIB[3].scParam2+3;
    myTIB[2].scParam2 = 1;

    myTIB[3].scOpcode = scNoInc;
    myTIB[3].scParam1 = (unsigned long) &Dummy;
    myTIB[3].scParam2 = 0;

    myTIB[4].scOpcode = scNoInc;
    myTIB[4].scParam1 = (unsigned long) &myTIB[6].scParam2;
    myTIB[4].scParam2 = 4;

    myTIB[5].scOpcode = scNoInc;
    myTIB[5].scParam1 = (unsigned long) &myTIB[7].scParam2;

```

```
myTIB[5].scParam2 = 4;

myTIB[6].scOpcode = scAdd;
myTIB[6].scParam1 = (unsigned long) &myTIB[1].scParam1;
myTIB[6].scParam2 = lwidth;

myTIB[7].scOpcode = scAdd;
myTIB[7].scParam1 = (unsigned long) &myTIB[8].scParam2;
myTIB[7].scParam2 = 0;

myTIB[8].scOpcode = scLoop;
myTIB[8].scParam1 = -70;
myTIB[8].scParam2 = Lines;

myTIB[9].scOpcode = scStop;
myTIB[9].scParam1 = (unsigned long) nil;
myTIB[9].scParam2 = (unsigned long) nil;

err = SCSIDataCommand(&mycmd, (SCSIInstr*)&myTIB, myGlobals.SCSI_ID, errors,
                      &stat, &message);
if (err<4){
    char s[256];
    sprintf(s,"Getlines SCSI Error count = %d",err);
    ParamText(CtoPstr(s), "\p","\p","\p");
    Alert(MsgAlertID, nil);
}

return err;
}

...
```

M.2.5 Extracts from Integ/Focus.c Source Listing

```

/*****
*
*   File      Integ/Focus.c
*
*   Contains Aperture and Focus functions for Treescan plugin
*
*****/

...

// -----
// Adjust integration time
// get a line and adjust integration time to make signal almost saturate...
// Passed:   Current integration time
// Returns:  Suggested new integration time
// -----
long DoAdjInt(DialogPtr myDialogPtr, int intsteps){
    long    x,i,startsteps,isteps,tsteps;
    long    total,average;
    Byte    max,*p;
    char    s[256];
    int     intID;

    if (CountDITL(myDialogPtr)< (dbBoxItemCount-10))
        intID = StatIntNoID;          // Main dialog
    else
        intID = db_intID;             // Debug dialog

    //Try and speed it getting a rough estimate of where to start

    // times 4 as best we can do is div by four
    for (tsteps = min_int_steps * 4; tsteps<max_int_steps; tsteps *= 10){
        if (GetChunk(1, 0, tsteps,0)<4){
            SysBeep(1); return intsteps;}
        if (pref.display) Display(0,1,myGlobals.height,1,pref.detail, myDialogPtr);
        max = total = 0;
        for(p = (Byte*)base + lwidth/2 - ewidth/2; p<((Byte*)base + lwidth/2 +
            ewidth/2); p++){
            total += *p;
            if (*p > max) max = *p;
        }
        average = total/ewidth;
        if (pref.AD10)
            sprintf((char*)s,"%lda* %ld", tsteps, average);
        else
            sprintf((char*)s,"%lda %ld", tsteps, average);
        SetDString(myDialogPtr, intID, CtoPstr(s));
        if (average>5) break;
        if ((!pref.AD10) && (tsteps>200)){
            pref.AD10 = true;
            tsteps /= 10; //ie try again with 10bit
        }
    }
    if (tsteps>max_int_steps){
        SysBeep(1); return max_int_steps;
    }
    startsteps = ((float)eaverage/average * tsteps)/2.0;

    for (isteps = startsteps; isteps < max_int_steps; isteps *= int_step_mult){
        if (GetChunk(1, 0, isteps,0)<4){
            SysBeep(1); return intsteps;}
        if (pref.display) Display(0,1,myGlobals.height,1,pref.detail, myDialogPtr);
        max = total = 0;
        for(p = (Byte*)base + lwidth/2 - ewidth/2; p<((Byte*)base + lwidth/2 +
            ewidth/2); p++){
            total += *p;
            if (*p > max) max = *p;
        }
        average = total/ewidth;
        if (pref.AD10)
            sprintf((char*)s,"%ldb* %ld", isteps, average);
        else
            sprintf((char*)s,"%ldb %ld", isteps, average);
        SetDString(myDialogPtr, intID, CtoPstr(s));
        if (average>eaverage) break;
    }

    if (!pref.AD10 && isteps>2000){
        pref.AD10 = true;
        for (isteps = startsteps/4; isteps < max_int_steps; isteps *= int_step_mult){

```

```

        if (GetChunk(1, 0, isteps,0)<4){
            SysBeep(1); return intsteps;}
        if (pref.display) Display(0,1,myGlobals.height,1,pref.detail,
            myDialogPtr);
        max = total = 0;
        for(p = (Byte*)base + lwidth/2 - ewidth/2; p<((Byte*)base + lwidth/2 +
            ewidth/2); p++){
            total += *p;
            if (*p > max) max = *p;
        }
        average = total/ewidth;
        sprintf((char*)s,"%ldc* %ld", isteps, average);
        SetDString(myDialogPtr, intID, CtoPstr(s));
        if (average>eaverage) break;
    }

    if (isteps>= max_int_steps){
        SysBeep(1);
        return max_int_steps;
    }else
        return isteps;
}

// -----
// New Adjust integration time
// Calculate average exposure for an area and guess new integration time
// Passed: Current integration time, number of lines already captured
// Returns: Suggested new integration time
// -----
long DoAdjInt2(DialogPtr myDialogPtr, long intsteps, int pp, short outrefNum){
    long x,i,j,isteps;
    long maxh,total,count;
    Byte max,av, *p;
    char s[100];
    int err;

    av = 0;
    isteps = intsteps;
    maxh = eheight;
    if (maxh){
        max = total = 0;
        for (j = (pp); j<pp+maxh; j++){
            for (p = (Byte*)(base + (long)lwidth*j) + lwidth/2 - ewidth/2;
                p<((Byte*)(base + (long)lwidth*j) + lwidth/2 + ewidth/2);
                p++){
                total += *p;
                if (*p > max) max = *p;
            }
        }
        av = (Byte)(total / maxh / ewidth);
        isteps = (float)eaverage/av * intsteps;
    }
    if (true && (isteps>2000) && !pref.AD10){
        isteps /= 4;
        pref.AD10 = true;
    }else if (true && (isteps<500) && pref.AD10){
        isteps *= 4;
        pref.AD10 = false;
    }

    if (pref.LogFiles){
        sprintf((char*)s,"%ld %d %d %ld %ld",
            total,max, av,intsteps,isteps);
        CtoPstr(s); count = s[0];
        err = FSWrite(outrefNum,&count,&s[1]);
        if (err) {SysBeep(2);FSClose(outrefNum);}
    }

    if ((isteps/intsteps<0.8) || (isteps/intsteps)>1.2)
        isteps = (isteps - intsteps) * 0.80 + intsteps;
    if (isteps>(intsteps*10)) isteps = intsteps/2;//try to trap silly
        saturation
    if (isteps>max_int_steps)
        isteps = max_int_steps;
    else if (isteps<min_int_steps){
        isteps = min_int_steps;
    }
}

if (pref.AD10)
    sprintf((char*)s,"%ld->%ld* %d", intsteps, isteps, av);
else
    sprintf((char*)s,"%ld->%ld %d", intsteps, isteps, av);
SetDString(myDialogPtr, StatAdjIntNoID, CtoPstr(s));
return isteps;
}

```

```

//*****
// Do auto Focus
// Decide whether the Mk1 or Mk2 routine should be used.
//*****
int DoFocus(DialogPtr myDialogPtr, sstatPtr sstat)
{
    if ( pref.Mk2 )
        return DoFocusMk2(myDialogPtr, sstat);
    else
        return DoFocusMk1(myDialogPtr, sstat);
}

//
//          ***** Do AutoFocus Mk2 (Using stepper) *****
//
// Return 1 if focused ok
int DoFocusMk2(DialogPtr myDialogPtr, sstatPtr sstat){
    int          a,b,i,j,mi,ni,mil;
    Byte         *p;
    char         s[256];
    double       mf,mfl;
    EventRecord   theEvent;
    long         count,x;
    OSErr        err;
    FSSpec       outfile;
    Str255       dname="\pFocus Data";
    short        outrefNum;
    int          fnumID, focusnID;

    if (CountDITL(myDialogPtr)< (dbBoxItemCount-10)){
        focusnID = StatFocusNo1ID; fnumID = StatFocusNo2ID;}           // Main dialog
    else{
        focusnID = db_focusnID; fnumID = db_fnumID;}                  //
        Debug dialog

    if (pref.LogFiles){
        strncpy(&s[1], (char*)&sstat->ViewName[1], sstat->ViewName[0]); s[0] = sstat->ViewName[0];
        PtoCstr((unsigned char *)s);
        if (sstat->viewnum==1)
            strcat(s, "_v1");
        else
            strcat(s, "_v2");
        sprintf((char*)dname, ":::Treescan Images:%s Focus Data", s);
        CtoPstr((char*)dname);
        err = FSPMakeFSSpec(0,0,dname,&outfile);
        err = FSpDelete(&outfile);
        err = FSpCreate(&outfile, 'Imag', 'TEXT', smSystemScript);
        if (err) {SysBeep(2);return 0;}

        err = FSpOpenDF(&outfile, fsCurPerm, &outrefNum);
        if (err) {SysBeep(2);return 0;}
    }

    DoMoveFocusInfinityMk2();    //focus at infinity
    mi = 0; mf = 0; ni = 0;
    for(i=0; i<f2gusteps; i+=f2bsteps){
        SetDNum(myDialogPtr, focusnID, fstepnum);
        if (GetChunk(1, 0, sstat->intsteps,0)<4){
            SysBeep(1); return 0;}
        calcfnum(s,&min,&max,&av);
        sprintf((char*)s, "> %f", focusnum);
        SetDString(myDialogPtr, fnumID, CtoPstr(s));
        if (pref.display) Display(0,1,myGlobals.height,1,pref.detail, myDialogPtr);
        if (pref.LogFiles){
            sprintf((char*)s, "%d %f %.0f %.0f %.0f %d %d %.1f\r",
                i, focusnum, focusnum2, focusnum4, focusnum8, min, max, av);
            CtoPstr(s); count = s[0];
            err = FSPWrite(outrefNum, &count, &s[1]);
            if (err) {SysBeep(2);FSClose(outrefNum);return 0;}
        }
        if (focusnum>mf){
            mi = i;
            mf = focusnum;
        }
        if (focusnum<(mf*0.6) && focusnum>1.0){
            break;
        }

        DoMoveFocusMk2(f2bsteps);
    }
    ni = i; //remember i steps reached

    mil = 0; mfl = 0;
    DoMoveFocusMk2((mi+f2bsteps)-ni);    // Move to almost max focus

```

```

for(i=(mi+f2bsteps); i>(mi-f2retsteps); i-=f2ssteps){
    if (i<0) break; // dont try and go back past infinity
    SetDNum(myDialogPtr, focusnID, fstepnum);
    if (GetChunk(1, 0, sstat->intsteps, 0)<4){
        SysBeep(1); return 0;}
    calcfnum(s, &min, &max, &av);
    sprintf((char*)s, "< %f", focusnum);
    SetDString(myDialogPtr, fnumID, CtoPstr(s));
    if (pref.display) Display(0,1,myGlobals.height,1,pref.detail, myDialogPtr);
    if (pref.LogFiles){
        sprintf((char*)s, "%d %f %f %f %f %d %d %f ssb\r",
            i, focusnum, focusnum2, focusnum4, focusnum8, min, max, av);
        CtoPstr(s); count = s[0];
        err = FSWrite(outrefNum, &count, &s[1]);
        if (err) {SysBeep(2); FSClose(outrefNum); return 0;}
    }
    if (focusnum>mfl){
        mil = i;
        mfl = focusnum;
    }
    DoMoveFocusMk2(-f2ssteps);
}

if (mfl>mf){
    DoMoveFocusMk2(mil-i); // move to mil max focus
    sprintf((char*)s, ".%d", (int)(mfl/100.0));
    SetDString(myDialogPtr, fnumID, CtoPstr(s));
    sprintf((char*)s, "%d %.0f fs-mil\r", mil, (float)(mfl));
}
else{
    DoMoveFocusMk2(mi-i); // move to mi max focus
    sprintf((char*)s, ".%d", (int)(mf/100.0));
    SetDString(myDialogPtr, fnumID, CtoPstr(s));
    sprintf((char*)s, "%d %.0f fs-mi\r", mi, (float)(mf));
}
SetDNum(myDialogPtr, focusnID, fstepnum);
if (GetChunk(1, 0, sstat->intsteps, 0)<4){
    SysBeep(1); return 0;}
calcfnum(s, &min, &max, &av);
sprintf((char*)s, ". %f", focusnum);
SetDString(myDialogPtr, fnumID, CtoPstr(s));
if (pref.display) Display(0,1,myGlobals.height,1,pref.detail, myDialogPtr);
if (pref.LogFiles){
    sprintf((char*)s, "%d %.0f %f %f %f %d %d %f final posn\r",
        i, focusnum, focusnum2, focusnum4, focusnum8, min, max, av);
    CtoPstr(s); count = s[0];
    err = FSWrite(outrefNum, &count, &s[1]);
    if (err) {SysBeep(2); FSClose(outrefNum); return 0;}
}
if (pref.LogFiles) FSClose(outrefNum);
return 1;
}

...

//
// ***** Do Blind Refocus (Using stepper) *****
//
// Use geometry to estimate actual distance and then use lookup table to adjust focus
// Return 1 as we assume we are still focussed
// step position
int DoBlindRefocus(DialogPtr myDialogPtr, int gotlines, float alpha, int Dist, float
    dip, int ys
    ,int *ftable, int dfirst, int dinc, int dlast, sstatPtr sstat){
    int a,b,i,j,mi;
    Byte *p;
    char s[256];
    double mf,last;
    EventRecord theEvent;
    long count,x;
    OSErr err;
    int wn,wstep;
    float o;

    o = (ys*alpha*myGlobals.StepAdj) + dip;
    wn = Dist/cos(o/180*3.14159);
    if ( wn <= dlast)
        wstep = ftable[(wn-dfirst)/dinc];
    else
        wstep = ftable[(dlast-dfirst)/dinc];
    /*{ Str255 s;
        sprintf((char*)s, "o = %f, wn = %d\r, wstep = %d fstepnum = %d", o, wn,
            wstep, fstepnum);
        CtoPstr((char *)s);
        ParamText(s, "\p", "\p ", "\p");
        Alert(MsgAlertID, nil);
    }*/

```



```
// fstepnum contains curent number of steps from infinity so if wstep is stenum I
// want
// then we should move wstep-fstepnum
DoMoveFocusMk2(wstep - fstepnum);
Delay(fdelay,&x);

if (pref.LogFiles){
    SetDNum(myDialogPtr,StatRefocusNo2ID,fstepnum);
    if (GetChunk(1, 0, sstat->intsteps,0)<4){
        SysBeep(1); return 0;}
    if (pref.display) Display(0,1,myGlobals.height,1,pref.detail, myDialogPtr);
    calcfnum(s,&min,&max,&av);
    SetDNum(myDialogPtr, StatRefocusNo1ID,wn);
}
return 1;
}
```

Appendix N

Image Processing Software

This appendix presents relevant sections of the image processing software. An overview of the code is provided with a breakdown into files. Listings are also provided of relevant sections of code. This includes the parameter extraction macros as well as the NIH Image additions.

- Parameter extraction macros (version 3.22)
- NIH Image modifications (version TF 3.5f)

N.1 Parameter Extraction Macros (Version 3.22)

The parameter extraction macros have been split into three files as a result of NIH Image only being able to load 32 kByte macro files.

Parameter extraction source files:

Image Macros	Main macro file which contains all the macros for normal processing. This includes calibration, height and diameter estimation, tree stem model generation, display of 3D stem model, and sweep estimation.
UTIL Macros	Utility macros which implement processing functions not part of the normal processing sequence. This includes printing, generation of thumbnail images, image feature removal, and filtering operations.
Modif History	TreeScan macros modification history.

This appendix contains a full listing of the Image Macros file.

N.1.1 Image Macros Source Listing

```

{  ** TreeScan ** image capture and image analysis macros
    Production Technology, Massey University, 1995.

    Version 2.23 (Note version number in 2 places in file please update both)

    Requires modified version of Image (version TF3.3)
        which includes modified user.p, image.rsrc & image.p

    For modification history see separate file.
        Computer specific : NIHPath,
        Requires : 'Background', 'TreeScan Help'
        Associated with utilities file
}

{***** global variables *****)
Var {image macros}
    b,d,ds,d2,d2s,a,as,dbds,d2sbds,xoffset:real;
    bcount,bzCount,grx5,grx5Orig,gry5,decimation:integer;
        alpha, realalpha,Crod: real;
    theta,tanTheta,cosTheta,sinTheta: real;
    cosasalpha, sinasalpha: real;
    dname, version, calibratedImage, image4model, modelImage:string;
    pi : real;
    Dist, Ht, W , dip, lean: real;
    Nkr, kr, mag : real;
    useo, mo, gotdip1, optionKey: boolean;
    progress : integer;
    NIHPath:string;

    {3D display global variables}
    x11,x12,x13,y1,iy,ix,x,y,pixstep,dmeas, scale:integer;
    SliceSize, TreeHeight, Slices: real;
    HS, VS, rotation, thi, theta, thetastep: real;
    bcount, minTreeHgt, maxTreeHgt:integer;
    ftheta: real;
    clearb4paint : boolean;
    minTreeHgt, maxTreeHgt: integer;

{***** procedures *****)
{-----}
procedure CheckOptionKey;
begin;
    SetCounter(7);
    rX[7] := 0;
    Usercode(6,1,2,3);
    if rX[7] = 1 then optionKey:=true else optionKey:=false;
end;
{-----}
procedure setPath;
begin;
    NIHPath := 'Tasman HD:Apps:NIH Image: ';
end;
{-----}
procedure LoadCalibDat;
begin;
    SetOptions('X-Y Center,User1,User2');
    SetCounter(7);
    rUser1[4] := 0; {d}
    Usercode(2,1,2,3);{Load calib data into measurements arrays}
    { if (rUser1[4]=0) then
        Beep
    else
        ShowResults;}
end;
{-----}
Procedure LoadDipLean;
begin;
    SetCounter(7);
    rX[7] := 0;
    Usercode(3,1,2,3);{Load dip and lean from the two data files and combine}
    if rX[7] = 1 then gotdip1:=true else gotdip1:=false;
    if gotdip1 = true then begin
        decimation := rY[7];
        dip := rUser1[5]/180*pi;
        lean := rUser1[6]/180*pi;
        ShowMessage('dip = ',dip/pi*180, '    lean = ',lean/pi*180)
    end else
        beep;

```

```

end;

{-----}
procedure RemovePerspective;
var
  otheta, xs, ys: real;
  temp, tx1, ty1, tx2, ty2, i, orCount, width, height, tempdec: integer;
  tempStr: string;
begin
  if rCount=0 then begin
    LoadCalibDat;
    if (rUser1[4]=0) then begin {No .calib file}
      PutMessage('No saved calibration data. Mark the left, centre, and
        right of the bottom calibration bar with the cross
        hair tool and then re-run this macro.');
```

ResetCounter;

exit;

end;

d2 := rUser1[1]; { actually we will pretend d2 = d and then scale d2s accordingly }

a := rUser1[2];

b := rUser1[3];

d := rUser1[4];

decimation := rY[7];

dip := rUser1[5]*pi/180;

lean := rUser1[6]*pi/180;

checkOptionKey;

if optionKey then begin

tempdec := GetNumber('Enter value for image decimation',1);

for i:=1 to rCount do

rY[i] := rY[i]/tempdec;

decimation:= decimation * tempdec;

end;

alpha := realalpha*decimation;

gry5 := rY[5];

grx5Orig := rX[5];

GetPicSize(width,height);

grx5 := width/2;

ds := rUser2[1];

as := rUser2[2];

theta := rUser2[3]*pi/180;

d2s := rUser2[4];

Dist := rUser2[5];

Ht := rUser2[6];

if rUser1[7] = 1 then mo := true

else mo:= false;

if rUser2[7] = 1 then useo := true

else useo:= false;

dbds := d/ds;

d2sbds := d2s/ds;

cosasalpha := cos(as*alpha);

sinasalpha := sin(as*alpha);

tantheta := (cosasalpha - d2sbds)/sinasalpha;

mag := d/ds;

W := mag/kr;

if useo=true then

if mo=true then

if gotdipl=true then begin

tempStr:='Calibration data loaded from file.\ - using O -

dip / lean from data file\';

end else begin

tempStr:='Calibration data loaded from file.\ - using O -

dip / lean manual entry\';

end else begin

tempStr:='Calibration data loaded from file.\ - using O from

modified O\';

end else begin

tempStr:='Calibration data loaded from file.\ - using A1

method\';

end;

ShowMessage(tempStr,'O =', theta/pi*180:2:1, ' D =', Dist:4:1, ' Ht =',Ht:4:1, ' Nkr =',Nkr:6:6, ' Mag =',Mag:6:4, ' W =',W:4:1, ' Dip = ',dip/pi*180, ' Lean = ', lean/pi*180, ' Decimation =', decimation);

end

else if (rCount<>3) and (rCount<>6) then begin

PutMessage('Mark 3 points on the bottom bar or 6 points on both bars of the calibration rod and then re-run this macro');

ResetCounter;

exit;

end

else begin

UserCode(1,1,2,3); {Enter real world calibration rod dimensions}

```

if rCount=3 then {must be entering 0 so just use bottom bar} begin
  mo := true; useo := true
end else begin
  mo := false;
  useo := false
end;
d2 := rUser1[1]; { actually we will pretend d2 = d and then scale d2s
                  accordingly }
a := rUser1[2];
b := rUser1[3];
d := rUser1[4];

If mo=true then begin
  SetCounter(6);
  rX[4]:=rX[1];
  rX[5]:=rX[2];
  rX[6]:=rX[3];
  rY[4]:=rY[1];
  rY[5]:=rY[2];
  rY[6]:=rY[3];
  rY[2]:=rY[2]+10; {just to stop divide by 0}
  a := 0; {so calculated y from these will all be b=140}
end;

gry5 := rY[5];
grx5 := rX[5];
rX[1]:=rX[1]-grx5; rY[1]:=rY[1]-gry5;
rX[2]:=rX[2]-grx5; rY[2]:=rY[2]-gry5;
rX[3]:=rX[3]-grx5; rY[3]:=rY[3]-gry5;
rX[4]:=rX[4]-grx5;
rX[6]:=rX[6]-grx5;

  checkOptionKey;
  LoadDipLean;
  if optionKey or gotdip=false then
    decimation := GetNumber('Enter value for steps per line',1);
    alpha := realalpha*decimation;

ds:= sqrt(sqr(rX[6] - rX[4]) + sqr(rY[6]-rY[4]));
as := rY[2];
d2s:= (sqrt(sqr(rX[3] - rX[1]) + sqr(rY[3]-rY[1])))*d/d2;
d2 := d;
dbds := d/ds;
d2sbds := d2s/ds;
cosasalpha := cos(as*alpha);
sinasalpha := sin(as*alpha);
tantheta := (cosasalpha - d2sbds)/sinasalpha;
theta := arctan(tantheta);
mag := d/ds;
Nkr := mag / 1500;
W := mag/kr;

if mo=true then begin
  if gotdip=false then begin
    decimation := GetNumber('Enter value for steps per line',0);
    dip := GetNumber('Enter value for dip in degrees',0)/180*pi;
    lean := GetNumber('Enter value for lean in degrees',0)/180*pi;
  end;
  theta := dip + lean;
end else begin
  otheta := theta;
  theta := GetNumber('Calculated value of O is',theta/pi*180)/180*pi;
  if (abs(theta-otheta)/pi*180>0.1) then
    useo := true;
end;
Dist := W * cos(theta);
Ht := W * sin(theta);

if useo=true then
  if mo=true then
    if gotdip=true then begin
      tempStr:='Calibration data calculated.\ - using O - dip / lean
               from data file\';
    end else begin
      tempStr:='Calibration data calculated.\ - using O - dip / lean
               manual entry\';
    end else begin
      tempStr:='Calibration data calculated.\ - using O from modified
               O\';
    end else begin
      tempStr:='Calibration data calculated.\ - using A1 method\';
    end;
  ShowMessage(tempStr,'O =', theta/pi*180:2:1, ' D =', Dist:4:1, ' Ht
    =',Ht:4:1,' Nkr =',Nkr:6:6,' Mag =',Mag:6:4,'
    W =',W:4:1, ' Dip = ',dip/pi*180, ' Lean = ',
    lean/pi*180, ' Decimation =', decimation);

```

```

(Save calibration stuff to a in measurements arrays and then store to a file)
setCounter(7);
setUser1Label('Calib Data');
setUser2Label('Calc Data');
rUser1[5] := dip/pi*180;
rUser1[6] := lean/pi*180;

rUser1[2] := a;
rUser2[1] := ds;
rUser2[2] := as;
rUser2[3] := theta/pi*180;
rUser2[4] := d2s;
rUser2[5] := Dist;
rUser2[6] := Ht;
rY[7] := decimation;
if mo=true then rUser1[7] := 1 else rUser1[7] := 0;
if useo=true then rUser2[7] := 1 else rUser2[7] := 0;
SetOptions('X-Y Center,User1,User2');
SetExport('Measurements');
Export(concat(WindowTitle,'.calib'));{try :2 to indicate dont put up dlog box}
end;
end;
{-----}
procedure CheckCalibration;
begin;
  if progress<>3 then begin
    PutMessage('Image has not been calibrated.      Run calibration macro:
               <F5>');
    exit;
  end;
  if (calibratedImage <> WindowTitle) then
    PutMessage('The calibration data for this image is not loaded
               ('.calibratedImage,' is loaded).      Calibrate
               image first using [F5].');
end;
{-----}
procedure setupText;
begin;
  SaveState;
  SetFont('Geneva');
  SetFontSize(9);
  SetText('Left');
end;
{-----}
procedure PlotTreeGrid;
var
  i:integer;
begin;
  if clearB4Paint then begin
    makeroi(x-50,y-3.5*pixstep,100,4*pixstep);
    clear;
  end;

  MoveTo(x,y);
  LineTo(x,y-3.5*pixstep);

  for i:= 0 to 3 do begin
    MoveTo(x-50,y-i*pixstep);
    LineTo(x+50,y-i*pixstep);
  end;
end;

{-----}
procedure Plot3DTree;
begin;
  rLength[1]:=x13;
  rLength[2]:=y1;
  Usercode(9,theta,HS,VS);
end;
{Plot a 3D tree}

{-----}
procedure plotscaleline;
var
  step:integer;
begin;
  MoveTo(x,y);
  LineTo(x,y+5);
  MoveTo(x-15,y+13);
  writeln(scale:3:0,'cm ');
end;
{-----}
procedure plotscale;
var
  step:real;
begin;
  scale:=50;

```



```

x:=x11+scale*HS; y:=y1;
plotscaleline;
x:=x12+scale*HS; y:=y1;
plotscaleline;
x:=x13+scale*HS; y:=y1;
plotscaleline;
scale:=-50;
x:=x11+scale*HS; y:=y1;
plotscaleline;
x:=x12+scale*HS; y:=y1;
plotscaleline;
x:=x13+scale*HS; y:=y1;
plotscaleline;
end;

{-----}
procedure cross;
begin;
    MoveTo(xs,ys-size);
    LineTo(xs,ys+size);

    MoveTo(xs-size,ys);
    LineTo(xs+size,ys);

end;

{-----}
procedure ModelInfo;
var
    DiamBH, SED, maxSwp, maxSwpHgt, maxSwpSED: real;
    temp:real;
    temp1, temp2 : string;
begin;
    maxTreeHgt:=rArea[1];
    minTreeHgt:=rMean[1];
    DiamBH:=rStdDev[1];
    SED:=rAngle[1];
    maxSwp:=rArea[2];
    maxSwpHgt:=rMean[2]/100;
    maxSwpSED:=rStdDev[2];
    slices := rUser1[1];
    sliceSize := rUser2[1];
    if (maxSwpSED/maxSwp<1) then begin
        temp1 := ''; temp2 := ' SED '; temp := maxSwp/maxSwpSED;
    end else begin
        temp1 := 'SED/'; temp2 := ''; temp := maxSwpSED/maxSwp;
    end ;

    ShowMessage('Stem model loaded.\\Max model Hgt = ', (maxTreeHgt/100):4:2, '
        m \\Min model Hgt = ', (minTreeHgt/100):4:2, ' m
        \\Diameter at breast Hgt = ',DiamBH:2:1,' cm \\Stem SED =
        ',SED,' cm \\Max sweep: ', temp1,temp2:2:0,temp2,' at
        height = ', maxSwpHgt:4:1,' m \\ Model consists of :
        ',slices:2:0,' slices,\\
        ',sliceSize:2:0,' cm.' );
    SelectWindow('Values');
end;

{***** Macros *****)
{-----}
macro 'TreeScan Help [F1]';
begin;
    setPath;
    Open(concat(NIHPath,'TreeScan Help'));
end;
{-----}
macro 'Image Capture(';
{-----}
macro '    Aquire Image from Treescan[F2]';
var
    test: string;
begin;
    test := WindowTitle;
    Acquire('Treescan');
    if test<>WindowTitle then
        save; {save if new image returned}
end;

{-----}
macro 'Parameter Extraction(';
{-----}
macro '    Load an image[F3]';
begin;
    Open('');
    progress :=1;
    resetCounter;
end;

```

```

{-----}
procedure F4MacroProcedure;
var
  tempH, tempW:integer;
begin;
  version := 'TreeScan Utility Macros\    version TF 2.23';
  setPath;

  ShowMessage(version);

  pi := 3.14159265;
  kr := 0.000175; {given mag in cm/pix and dist in cm}
  realalpha := 0.0103021978 * pi / 180; {0.010332}
  dname := WindowTitle;
  setOptions('');
  InvertY(1);
  killRoi;
  measure;
  If (histogram[0])>0 then begin
    AddConstant(1);
  end;
  SetForegroundColour(255);
  SetScale(0,'pixels');
  ResetCounter;

  bCount := 0;
  bzCount := 0;

  GetPicSize(tempW,tempH);
  ShowMessage(version);
  progress := 2;
end;

macro '    Remove white pixels[F4]';
{ Removes white pixels by adding 1 to all grey levels if necessary
  then sets foreground to black. Should then set correct tool for marking
  6 points on the calibration rod.
}
begin
  F4MacroProcedure;
end;

{-----}
macro '    Perspective Calibration[F5]';
{ Assumes calibration data saved on disk}
begin;
  if progress<>2 then
    if rCount=0 then begin
      PutMessage('Remove white pixels using F4 macro first.');
```

exit;

```
    end else begin
      ResetCounter;
      F4MacroProcedure;
    end;
  ShowMessage(version);
  RemovePerspective;
  xoffset := (grx5Orig- grx5)*dbds*cos(theta)/cos(theta+0*alpha);
  setUser1Label('X in cm');
  setUser2Label('Y in cm');
  setPrecision(1);
  setScale(0,'pixels');
  SetForegroundColour(0);
  ResetCounter;
  calibratedImage := WindowTitle;
  progress := 3; { Image Calibrated for Perspective}
end;

{-----}
macro '    Tree edges to data points [F6]';
{
  Requires an image of a tree trunk with the edges of the trunk marked in white
}
var
  w,h,xs,ys,yint,left,top,width,height,i,ii,nPixels,mean,mode,min,
  max:integer;
  lasty, lastx,x,y:real;
  pointsThisLine,spacing:integer;
begin
  CheckCalibration;
  RequiresVersion(1.45);
  Measure;
  GetResults(nPixels,mean,mode,min,max);
  if (histogram[0])<100 then begin
    PutMessage('First mark the edges in white');
```

exit;

```
  end;
  SaveState;

```

```

GetRoi(left,top,width,height);
GetPicSize(w,h);
if width=0 then begin
  SelectAll;
  GetRoi(left,top,width,height);
end;
Duplicate('Tree Edges');
Invert;
SetForegroundColor(0);
SetLineWidth(1);
SetThreshold(255);
checkOptionKey;
spacing := 10;
if optionKey then
  spacing:=getNumber('Measurement spacing (pixels):',2);
  i:=1; ii:=1;
repeat
  MoveTo(0,i); LineTo(width,i);
  i:=i+1;
  ii:=ii+1;
  if ii=spacing then begin
    ii:=1;
    i:=i+1;
  end;
until i>height;
setUser1Label('X in cm');
setUser2Label('Y in cm');
setPrecision(1);
setOptions('Area,User1,User2');
LabelParticles(false);
SetParticleSize(1,999999);
AnalyzeParticles;
ii:=1;
lasty:=0;
pointsThisLine:=0;
for i:=1 to rCount do begin
  rX[i] := rX[i] + left;
  rY[i] := rY[i] + h- top - height;
  xs := rX[i]-grx5;
  ys := rY[i]-gry5;
  if useo=true then
    y :=sin(theta + ys * alpha)/cos(theta + ys * alpha)*Dist- Ht+b
  else
    y := a*cos(theta+as*alpha)/sinasalpha*sin(ys*alpha)/cos(theta+ys*alpha)
      + b;
  x := xs*dbds*cos(theta)/cos(theta+ys*alpha)-xoffset;
  if (pointsThisLine=0) and (y<>lasty) then begin
    rArea[ii]:=y;
    rUser1[ii]:=x;
    pointsThisLine:=1;
    {accept first point}
  end else if pointsThisLine=1 then begin
    if y=lasty then begin
      if (x-lastx)>10 then begin {provided >10cm apart -> record
        as pt2}
        rUser2[ii]:=x;
        ii:=ii+1;
        pointsThisLine:=0;
      end;
    end else begin
      rArea[ii]:=y;
      first point;
      rUser1[ii]:=x;
      pointsThisLine:=1;
    end;
  end;
  lastx:=x;
  lasty:=y;
end;
RestoreState;
SetCounter(ii-1);
Dispose;
ShowResults;
SetExport('Measurements');
Export(concat(dname,'.stem'));
end;
}-----}
macro ' Do 3D file conversion[F7]';
var
spacing : real;
begin;
{
  progress:=5;
  SetCounter(7);
  rX[7] := 0;
  Usercode(7,1,2,3);
  if (rX[7]=0) then begin {Load 2 .stem files and combine}

```

```

        Beep;
        PutMessage('Edge files not found. Please ensure both views have been
                    processed and one of the views is selected.');
```

end else begin

```

        PutMessage('3D conversion completed. Model in memory and saved to disk.');
```

{ ShowResults; }

```

        ModelInfo;
                                image4model := calibratedImage;
        progress:=6;
    end;
end;
```

-----}

```

macro '      Display 3D model [F8]';
var
    lasti,i,ii: integer;
    tempXAv, tempXD, tempZAv, temp2D:real;
    firstpoint:boolean;
begin;
    if progress<>6 then begin
        PutMessage('Load 3D model first.');
```

exit;

```

    end;

    xl1:=150; xl2:=250; xl3:=400; yl:=400; iy:=450; ix:=550; pixstep:=100;
    pi:=3.1416; thetastep:=45;
    SliceSize:=rUser2[1]; Slices:=rUser1[1]; TreeHeight:=Slices*SliceSize;
    HS := 0.5; {Pixels/cm horizontal}
    VS:= (4*pixstep/4000)*SliceSize ; {Pixels/slice vertical (pix/cm*stepsize)}
    Open(concat(NIHPath,'Background'));

    plotScale;

    {Plot 1st view of the tree}
    i:=rCount;
    firstpoint:=true;
    for ii:=0 to (Slices-1) do begin
        if firstpoint then MoveTo(xl1+(rUser1[i]+rMean[i]/2)*HS,yl-ii*VS);
        if rAngle[i]=1 then begin
            LineTo(xl1+(rUser1[i]-rMean[i]/2)*HS,yl-ii*VS);
            firstPoint:=false;
            lasti:=i;
        end;
        i:=i-1;
    end;
    LineTo(xl1+((rUser1[lasti]+rMean[lasti]/2)*HS),yl-(rCount-lasti)*VS);
    i:=rCount;
    firstpoint:=true;
    for ii:=0 to (Slices-1) do begin
        if firstpoint then MoveTo(xl1+(rUser1[i]+rMean[i]/2)*HS,yl-ii*VS);
        if rAngle[i]=1 then begin
            LineTo(xl1+(rUser1[i]+rMean[i]/2)*HS,yl-ii*VS);
            firstPoint:=false;
        end;
        i:=i-1;
    end;

    {Plot 2ndview of the tree}
    i:=rCount;
    firstpoint:=true;
    for ii:=0 to (Slices-1) do begin
        if firstpoint then MoveTo(xl2+((rUser2[i]+rStdDev[i]/2)*HS),yl-ii*VS);
        if rAngle[i]=1 then begin
            LineTo(xl2+((rUser2[i]-rStdDev[i]/2)*HS),yl-ii*VS);
            firstPoint:=false;
            lasti:=i;
        end;
        i:=i-1;
    end;
    LineTo(xl2+((rUser2[lasti]+rStdDev[lasti]/2)*HS),yl-(rCount-lasti)*VS);
    i:=rCount;
    firstpoint:=true;
    for ii:=0 to (Slices-1) do begin
        if firstpoint then MoveTo(xl2+((rUser2[i]+rStdDev[i]/2)*HS),yl-ii*VS);
        if rAngle[i]=1 then begin
            LineTo(xl2+((rUser2[i]+rStdDev[i]/2)*HS),yl-ii*VS);
            firstPoint:=false;
            lasti:=i;
        end;
        i:=i-1;
    end;

    theta:=0;
    Plot3DTree;
    progress:=7;
    saveas(concat(image4model,'.3D pic'));
    modelImage := WindowTitle;
```

```

end;
{-----}
macro '      (';
{-----}
macro '      Display x,y posn in cm[F9]'
var
  test, x,y,yn,xs,ys:real;
  width,height:integer;
begin;
  CheckCalibration;

  GetPicSize(width,height);
  GetMouse(xs,ys);
  checkOptionKey;
  if optionKey then begin
    MoveTo(xs-2,ys-2); LineTo(xs+2,ys+2);
    MoveTo(xs-2,ys+2); LineTo(xs+2,ys-2); MoveTo(xs+10,ys);
  end;

  ys :=height-ys-1;
  xs := (xs- grx5);
  ys := ys-gry5;
(PutMessage('pix >/\:xs:', xs , ' ys:' , ys));

  x := xs*dbds*cos(theta)/cos(theta+ys*alpha)-xoffset;
  y := a*cos(theta+as*alpha)/sinasalpha*sin(ys*alpha)/cos(theta+ys*alpha) + b;
  yn :=sin(theta + ys * alpha)/cos(theta + ys * alpha)*Dist- Ht+b;
  ShowMessage('Height\      = ',(yn/100):3:2,' m\Horizontal offset\      =
    ',x:0:1,' cm ');

  { ShowMessage(' (X,Yal,Yo) \(' ,x:0:1,' , ' ,y:3:1,' , ' ,yn:3:1,' )cm');
  { theta := GetNumber(' O is',theta/pi*180)/180*pi; }
  if optionKey then begin
    setupText;
    if useo=true then begin
      writeln('(',x:0:1,' , ' ,yn:3:1,' )');
    end else begin
      writeln('(',x:0:1,' , ' ,y:3:1,' )');
    end;
    restoreState;
  end;
end;

{-----}
macro '      Display distance in cm[F10]'
var
  dia,hei,x2s,y2s,x1s,y1s,yint:real;
  x1,x2,y1,y2,top,left,height:integer;
  width,height:integer;
begin;
  CheckCalibration;
  GetLine(x1s,y1s,x2s,y2s,width);
  if x1s<0 then begin
    PutMessage('This macro requires a line selection. ');
    exit;
  end;

  setForegroundColour(1);
  checkOptionKey;
  if optionKey then begin
    MoveTo(x1s,y1s); LineTo(x2s,y2s); MoveTo(x2s+10,y2s);
  end;
  GetPicSize(width,height);

  y1s :=height-y1s-1;
  x1s := x1s- grx5 ;
  y1s := y1s-gry5;
  x1 := x1s*dbds*cos(theta)/cos(theta+y1s*alpha)-xoffset;
  if useo=true then
    y1 :=sin(theta + y1s * alpha)/cos(theta + y1s * alpha)*Dist- Ht+b
  else
    y1 := a*cos(theta+as*alpha)/sinasalpha*sin(y1s*alpha)/cos(theta+y1s*alpha)
      + b;

  y2s :=height-y2s-1;
  x2s := (x2s- grx5) ;
  y2s := y2s-gry5;
  x2 := x2s*dbds*cos(theta)/cos(theta+y2s*alpha)-xoffset;
  if useo=true then
    y2 :=sin(theta + y2s * alpha)/cos(theta + y2s * alpha)*Dist- Ht+b
  else
    y2 := a*cos(theta+as*alpha)/sinasalpha*sin(y2s*alpha)/cos(theta+y2s*alpha)
      + b;

  dia := sqrt(sqr(x1-x2) + sqr(y1-y2));
  hei := (y1+y2)/2;

```

```

ShowMessage('Diameter\          = ',dia:0:1,' cm',chr(13),'Av height\          = ',hei/100:0:2,' m');
if optionKey then begin
  setupText;
  writeln(dia:0:1,' @ ',hei:0:1);
  restoreState;
end;
setForegroundColour(0);
end;

{-----}
macro '      Display sweep[F11]'
var
  height, sweeplow, sweephigh, sweepHgt, temp: integer;
  temp1, temp2:string;
  xls, yls, x2s, y2s, width: integer;
begin;
  if progress<>7 then begin
    PutMessage('Display 3D model first. ');
    exit;
  end;
  GetLine(xls,yls,x2s,y2s,width);
  if (xls>=0) then begin
    if yls=y2s then yls:=y2s+1;
    if (yls>y2s) then begin
      sweeplow:=(yl-yls)/VS*SliceSize;
      sweephigh:=(yl-y2s)/VS*SliceSize;
    end else begin
      sweeplow:=(yl-y2s)/VS*SliceSize;
      sweephigh:=(yl-yls)/VS*SliceSize;
    end;

    sweepHgt := sweephigh - sweeplow;

  end else begin
    sweepHgt := 600;
    GetMouse(xls,yls);
    sweeplow:=(yl-yls)/VS*SliceSize - sweepHgt/2;
    sweephigh:=(yl-yls)/VS*SliceSize + sweepHgt/2;
    MakeLineROI(xls,yls-(sweepHgt/2/sliceSize*VS),xls,
               yls+(sweepHgt/2/sliceSize*VS));
  end;

  if (sweeplow<minTreeHgt) then begin
    sweeplow := minTreeHgt;
    if ((sweeplow+sweepHgt)<maxTreeHgt) then
      sweephigh := minTreeHgt + sweepHgt;
  end;

  if (sweephigh>maxTreeHgt) then begin
    sweephigh := maxTreeHgt;
    if ((sweephigh-sweepHgt)>minTreeHgt) then
      sweeplow := maxTreeHgt - sweepHgt;
  end;

  Usercode(5,sweeplow, sweephigh, rUser2[1]);      {Calculate sweep}
  if ( rX[2]/rX[3]<1) then begin
    temp1 := ''; temp2 := ' SED '; temp := rX[3]/rX[2];
  end else begin
    temp1 := 'SED/'; temp2 := ''; temp := rX[2]/rX[3];
  end;
  ShowMessage('Sweep of ',temp1,temp:2:1,temp2,' over ',rX[1]/100:1:0,' m
    section.\ \ Maximum sweep = ',rX[3],' cm\ at height = ',
    rX[4]/100:3:1,' m\ and sed = ',rX[2]:3:0,' cm \ \Section
    max height = ', sweephigh/100:2:1,' m\Section min height
    = ', sweeplow/100:2:1,' m' );
  SelectWindow('Values');
end;

{-----}
macro '      Mark Scale[F12]'
var
  yi,xi,xs,ys,ysi,xsi,,yz,yint, xscale, yscale:real;
  OTH, OrgW, OrgH, NewW, NewH, xo, yo,x,y:integer;
  OrgPic, NewPic, Pix, RefH ,N, size: integer;
begin;
  CheckCalibration;
  N:=1; size:=10;
  GetPicSize(OrgW,OrgH);
  for yo:=0 to 40 do
    begin
      for xo:=-N to N do
        begin
          ysi := (arctan(((yo*100)+Ht-b)/Dist)- theta)/alpha;
          xsi := (xo*100+xoffset)*cos(theta +ysi*alpha)/(dbds*cos(theta));

          xs := xsi+grx5;

```

```

ys:= OrgH-(ysi+1+gry5);

if ((Round(yo/10)=yo/10) and (xo=0)) then
begin
    size:=20;
    MoveTo(xs+25,ys); writeln(yo,' m' );
    MoveTo(xs-50,ys); writeln(yo,' m' );
end
else if ((Round(yo/5)=yo/5) and ( xo=0)) then
begin
    size:=10;
    MoveTo(xs-50,ys); writeln(yo,' m' );
    MoveTo(xs+25,ys); writeln(yo,' m' );
end
else size:=5;

cross;
ShowMessage('(x,y) \('',xi:0:1,'','yi:3:1,'')');
end;
end;

{***** Additional macros *****)
{-----}
macro 'Utilities';
{-----}
macro '    Load 3D file information[8]';
var
spacing : real;
begin;
    setPath;
    SetCounter(7);
    rX[7] := 0;
    Usercode(8,1,2,3);                {Load .3D file}
    if (rX[7]=0) then begin
        Beep;
        PutMessage('3D model file not found. Ensure 3D conversion has been
                    completed and one of the views is selected.');
```

end else begin
 image4model := WindowTitle;
 ModelInfo;
 progress:=6;
end;
end;

```

{-----}
macro '    Rotate 3D model right [ ]';
var
    i,ii: integer;
    tempXAv, tempXD, tempZAv, tempZD:real;
begin;
    if (modelImage <> WindowTitle ) then
    begin
        Usercode(4,4,2,3);
        exit;
    end;
    theta:=theta-thetastep*pi/180;

    clearB4Paint:=true;
    x:=xl3; y:=yl;
    PlotTreeGrid;
    MoveTo(x,y);
    LineTo(x-30,y+30);

    Plot3DTree;
    plotScale;
end;
{-----}
macro '    Rotate 3D model left [ ]';
var
    i,ii: integer;
    tempXAv, tempXD, tempZAv, tempZD:real;
begin;
    if (modelImage <> WindowTitle ) then
    begin
        Usercode(4,3,2,3);
        exit;
    end;
    theta:=theta+thetastep*pi/180;

    clearB4Paint:=true;
    x:=xl3; y:=yl;
    PlotTreeGrid;
    MoveTo(x,y);
    LineTo(x-30,y+30);

    Plot3DTree;
    plotScale;
```



```
end;
{-----}
macro '      ScrollUp[-]';
begin;
  Usercode(4,1,2,3);
end;

{-----}
macro '      ScrollDown[]';
begin;
  Usercode(4,2,2,3);
end;

{-----}
macro '      Deselect [ ]'      {esc}
begin;
  killROI;
end;

{-----}
macro '      Load processing macros/M';
begin;
  setPath;
  Open(concat(NIHPath,'UTIL Macros'));
  ShowMessage('Loading:\      TreeScan Utility Macros');
  Usercode(10,1,2,3);
end;
```

N.2 NIH Image Additions (Version TF 3.5f)

The NIH Image source files are set up in such a way that a programmer can add their own routines to the User.p file. These code routines can then be called from the macro language using the Usercode() call. Ten routines have been added to the User.p file for the TreeScan system.

NIH Image Additions source files

User.p File to which pascal user routines may be added to NIH Image to speed up or add additional capabilities to the NIH Image macro language.

This appendix contains several relevant sections out of the User.p file.

N.2.1 User.p Source Listing

```

unit User;

{Modification History}
{30/1/95 RHP    V2.0 for Tasman}
{
  }
{
  Add a user routine to load calib data from a file )
  Image.calib which is written by the calibration macro}
{ 31/1/95 RHP  V2.1 Allow for calibration rod with unequal bars}
{
  load calib data file now doesnt show file chooser dlog}
{   1/2/95 RHP  V2.2 Load data file with lean and dip angles in them}
{
  modified plugins.p to try and get image name set in}
{
  plugin back into image}
{
  V2.3 }
{
  V2.4 }
{
  V2.5 Copy the vref from acquire plugin so we save image to correct
  folder}
{
  V2.7 when loading .dat files with dip and lean use file name uip to
  v1 or v2; }
{
  user7 routine which returns 1 if the option key is down}
{   9/3/95 MW  V2.8 2D to 3D file conversion, load 3D information}
{
  V2.9 Plot 3D tree routine}
{
  V3.0 Modified calib rod default to 200.55cm}
{ 20/9/95 MW  V3.1 Added .3DO output file format}
{ 11/10/95 MW V3.2 Modified .3DO output file format(LF->CR/LF, ! on pcode no's)}
{ 12/10/95 RHP V3.3 We didnt put buffers back if there was an error - fixed}
{
  V3.3a On call plugin tell them they can use all space including cut &
  undo buf not/2}
{ 16/10/95 MW V3.4 Sideways arrows, .3D additional info, calculate sweep &
  maxsweep, several other updates}
{ 24/10/95 MW V3.5 Minor 3DO output format modifications}
{   8/11/95 MW V3.5b 3D display position line fix }
{ 16/11/95 MW V3.5c User.p code to load macros from within a macro }
{ 17/11/95 MW V3.5d Remove buffer size checking in LineROI (Image.p) }
{ 20/11/95 MW V3.5f Sorting the loaded branch data (Dev1)}

...

procedure RHPCalibrod;
var
  itemhit, mydialogid: integer;
  pmydialog: DialogPtr;

begin
  mydialogid := 129;
  pmydialog := GetNewDialog(mydialogid, nil, POINTER(-1));
  if (pmydialog <> nil) then begin
    SetPort(pmydialog);
    ShowWindow(pmydialog);
    SelIText(pmydialog, 3, 0, 32767);
    repeat
      ModalDialog(nil, itemhit);
      if itemhit = 4 then
        itemhit := 4;
    until (itemhit = ok) or (itemhit = cancel);
    User1^[1] := GetDReal(pmydialog, 3);
    User1^[2] := GetDReal(pmydialog, 4);
    User1^[3] := GetDReal(pmydialog, 5);
    User1^[4] := GetDReal(pmydialog, 6);
    if itemhit = cancel then begin
      DisposDialog(pmydialog);
      exit(RHPCalibrod);
    end;
    DisposDialog(pmydialog);
  end;
end;

procedure RHPLoadCalibData;
var
  fname: str255;
  err: OSErr;
  RefNum, nValues, i: integer;
  rLine: RealLine;
  FinderInfo: FInfo;
begin
  fname := concat(Info^.title, '.calib');
  RefNum := Info^.vref;
  if not (GetFInfo(fname, RefNum, FinderInfo) = noerr) then
    exit(RHPLoadCalibData);
  ShowMessage(concat('Loading from ', fname));
  InitTextInput(fname, RefNum);

```

```

i := 1;
while not TextEOF do begin
  GetLineFromText(rLine, nValues);
  xcenter^[i] := rLine[1];
  ycenter^[i] := rLine[2];
  User1^[i] := rLine[3];
  User2^[i] := rLine[4];
  i := i + 1;
end;
end;

procedure MWScroll (Param1: extended);
var
  DeltaH, DeltaV, width, height, ScrollDirection: integer;
  loc: point;
  SaveSR: rect;
  WasDigitizing: boolean;
begin
  with info^ do begin
    if ScaleToFitWindow then begin
      PutMessage('Scrolling does not work in "Scale to Fit Window" mode.');
```

{ Up }

{ Down }

{ Left }

{ Right }

```

      exit(MWScroll)
    end;

    ScrollDirection := round(param1);
    with SrcRect do begin
      width := right - left;
      height := bottom - top;
    end;
    SaveSR := SrcRect;

    DeltaH := 0;
    DeltaV := 0;

    case ScrollDirection of
      1:
        DeltaV := round(-height * 0.8);
      2:
        DeltaV := round(height * 0.8);
      3:
        DeltaH := round(-width * 0.5);
      4:
        DeltaH := round(width * 0.5);
    otherwise
      ShowNoCodeMessage;
    end;

    with SrcRect do begin
      left := SaveSR.left + DeltaH;
      top := SaveSR.top + DeltaV;

      if OptionKeyDown and ((ScrollDirection = 1) or (ScrollDirection = 2)) then
        begin
          left := (PicRect.right - PicRect.left) div 2 - width div 2;
          right := (PicRect.right - PicRect.left) div 2 + width div 2;
          end;

          if OptionKeyDown then
            case ScrollDirection of
              1:
                top := PicRect.top;
              2:
                top := PicRect.bottom - height;
              3:
                left := PicRect.left;
              4:
                left := PicRect.right - height;
            end;

            if left < 0 then
              left := 0;
              if (left + width) > PicRect.right then
                left := PicRect.right - width;
                right := left + width;

              if top < 0 then
                top := 0;
                if (top + height) > PicRect.bottom then
                  top := PicRect.bottom - height;
                  bottom := top + height;

            end;
            UpdatePicWindow;
            DrawMyGrowIcon(wptr);

            WhatToUndo := NothingToUndo;

```

```

    ShowRoi;
    end; {with info^}
end;

...

procedure MeasMaxSweep (Param1, Param2: extended);
var
    distance, slice, loop, height1, height2: integer;
    TDmaxSweep, TDmaxSweepHgt, TDmaxSweepSED: real;
    temp1, temp2, temp3, temp4: str255;
begin
    height1 := round(Param1);
    height2 := round(Param2);
    distance := 600;
    slice := round(User2^[1]);
    TDmaxSweep := 0;

    if (height2 - height1) > distance then begin
        for loop := height1 to (height2 - distance) do begin
            MWMearSweep(loop, loop + distance, slice);
            if (xcenter^[3] > TDmaxSweep) then begin

                TDmaxSweep := xcenter^[3];           { Max sweep in cm }
                TDmaxSweepHgt := xcenter^[4];         { Height of max sweep }
                TDmaxSweepSED := xcenter^[2];         { Height of max sweep }
            end;
            loop := loop + slice;
        end;
        xcenter^[1] := distance;                     { Distance of sweep measurement in cm }
        xcenter^[2] := TDmaxSweepSED;                 { SED }
        xcenter^[3] := TDmaxSweep;                    { Max sweep in cm }
        xcenter^[4] := TDmaxSweepHgt;                 { Height of max sweep }
    end;
end;

...

procedure MWPlot3DTree (Param1, Param2, Param3: extended);
var
    i, ii: integer;
    tempXD, tempZD, tempXAv, tempZAv: extended;
    theta, HS, VS: extended;
    left, top, width, height, xl3, yl: integer;
    p1, p2: point;
begin
    {Plot 3D view of the tree}
    i := mCount;
    theta := param1;
    HS := param2;
    VS := param3;
    xl3 := round(plength^[1]);
    yl := round(plength^[2]);

    CurrentX := xl3 - 2;                             {MoveTo ...}
    CurrentY := yl - 2;
    tempXAv := 0 * cos(theta) + (-70) * sin(theta);
    tempZAv := -0 * sin(theta) + (-70) * cos(theta);

    LineWidth := 5;
    MWLineTo(round(xl3 - 2 + (tempXAv + tempZAv * 0.707) * HS), round(yl - 2 + (-tempZAv * 0.707) * HS));
    LineWidth := 1;

    for ii := 1 to (mCount - 1) do begin
        if orientation^[i] = 1 then                    {If slice has valid diameters}
        begin
            tempXD := mean^[i];
            tempZD := sd^[i];

            tempXAv := User1^[i] * cos(theta) - User2^[i] * sin(theta);
            tempZAv := -User1^[i] * sin(theta) + User2^[i] * cos(theta);

            with Info^ do begin
                {MakeOvalRoi command}
                RoiType := OvalRoi;
                left := round(xl3 + (tempXAv + tempZAv * 0.707 - tempXD / 2) * HS);
                top := round(yl - ii * VS + (-tempZAv * 0.707 - tempZD / 2 * 0.707) * HS);
                width := round(tempXD * HS);
                height := round(tempZD * 0.707 * HS);
                SetRect(RoiRect, left, top, left + width, top + height);
                MakeRegion;
            end;

            SetForegroundColor(0);
            DoOperation(PaintOp);
        end;
    end;
end;

```

```

SetForegroundColor(255);
DoOperation(FrameOp);
with Info^ do begin
  UpdateScreen(RoiRect);
end;
if (mArea^[i] = 500) or (mArea^[i] = 1000) or (mArea^[i] = 1500) or
(mArea^[i] = 2000) or (mArea^[i] = 2500) then begin
  CurrentX := x13;
  CurrentY := round(y1 - (ii - 2) * VS);
  MoveTo ...;
  MWLineTo(round(x13 + tempXAv * HS), round(y1 - (ii - 2) * VS));
  MWLineTo(round(x13 + (tempXAv + tempZAv * 0.707) * HS), round(y1 - (ii -
2) * VS + (-tempZAv * 0.707) * HS));
  MWLineTo(round(x13 + (tempZAv * 0.707) * HS), round(y1 - (ii - 2) * VS +
(-tempZAv * 0.707) * HS));
  MWLineTo(round(x13), round(y1 - (ii - 2) * VS));
end;

end;
i := i - 1;
end;
end;

...

procedure OldUserMacroCode (CodeNumber: integer; Param1, Param2, Param3:
extended);
begin
  case CodeNumber of
    1:
      RHPCalibrod;
    2:
      RHPLoadCalibData;
    3:
      RHPLoadDipLean;
    4:
      MWScroll(Param1);
    5:
      MWMeasSweep(Param1, Param2, Param3);
    6:
      MWCheckOptionKey;
    7:
      MWConvert3D;
    8:
      MWLoad3DData;
    9:
      MWPlot3DTree(Param1, Param2, Param3);
    10:
      MWLoadMacros;
    otherwise
      ShowNoCodeMessage;
  end;
end;
end;

```