TreeScanV and Frame Mosaicing

Thesis submitted as partial fulfilment of the requirements of the

Masterate Degree in Information Engineering

By: Farshad Nourozi
February 1997

Department of Production Technology
Massey University

ABSTRACT

In 1993 the Department of Production Technology carried out a feasibility study of applying digital imaging technology in the pre-harvest inventory assessment for the forestry industry. Consequently a scanning mechanism was developed to capture a series of overlapping images along the stem of a tree. These overlapping images needed to be registered and combined to form a single long and thin high resolution image of the tree.

This report describes different methods of finding the overlaps between the consecutive images. Algorithms developed here fall into two broad categories: Spatial Domain and Frequency Domain feature matching. Comparison of different algorithms is made and advantages and disadvantages of each one are discussed. Finally a robust algorithm is developed which combines the strengths of the other algorithms.

ACKNOWLEDGMENTS

If it were not for the encouragement of my dear wife, Fariba, I would not have started the project. Moreover, if it were not for her sacrificial support during the course of this project, I could not have finished it by now. My sincere thanks to her.

I would also like to thank all the people in the Department of Production Technology who helped me in one way or another in developing this system. In particular I would like to thank my supervisors Professor Bob Hodgson and Dr. Ralph Pugmire for providing direction in this development.

I would also like to thank Swidbert who was working on a similar project for making his work available to the image processing community on the internet.

TABLE OF CONTENTS

1- INTRODUCTION

A two dimensional grey scale digital image consists of a matrix of numbers often represented by 8 bits each. Each number represents one picture element or pixel, each pixel has one of 256 possible grey levels. The resolution depth of the image is defined by the number of bits in a pixel or the grey scale quantisation. The spatial resolution of the image is defined by the number of pixels in each row and column of the image matrix.

When a digital image of an object is used for measurement of the dimensions of the object, one of the factors that limits the resolution of the measurement is the spatial resolution of the image. As a rule of thumb, in simple image measurements one pixels is required per smallest unit of measurement. However there are interpolation techniques which allow measurement to sub-pixel resolution. These techniques rely on the fact that each pixel, although does not show every detail of the underlaying object, it has mixed and integrated those fine details into a lump. Therefore some information about the underlaying details may be extracted from the lump .

In 1993 the Department of Production Technology carried out a feasibility study for Tasman Forestry to determine possible ways of capturing high resolution images of standing trees in forests for tree feature measurement. The required accuracy on the measurements from the stem of the tree was ±1cm. A typical tree has a stem diameter of about a meter and a height of about 40 meters. At that time it was decided that an image resolution of about 500×8000 was needed to easily provide the required accuracy.

Among other possible methods, using a small video tape camera to capture a series of images with some overlap from sections of the tree (TreeScanV) was proposed. To obtain a high resolution image, these individual low resolution images had to be combined to form a suitably high resolution image (See Figure 1.1).

This method was implemented in two phases. One involved the design and construction of a scanning system to capture a series of overlapping images and transfering them to a personal computer. This work was carried out in 1995 and the details of its implementation are included in appendix 1. Phase two of the implementation involved analysing the low resolution images obtained in phase

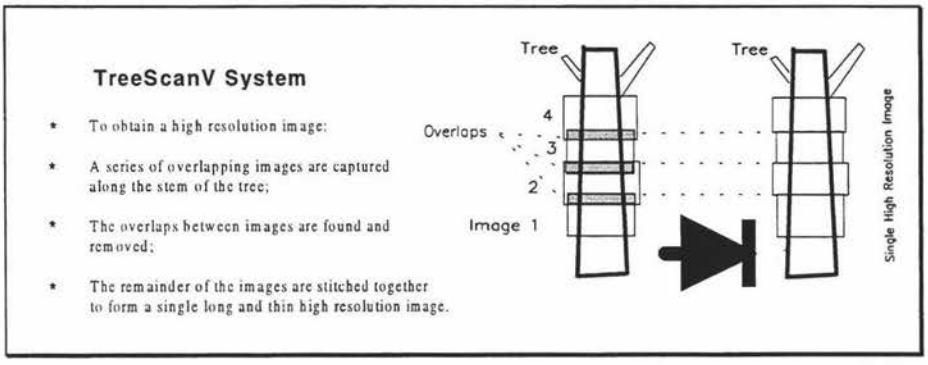


Figure 1.1- TreeScan

one and producing a single high resolution image. This report describes phase two of the implementation.

2- SYSTEM OVERVIEW

Considering the top level representation of the system as shown in figure 2.1, the objective of the system is to process the input images in order to produce the output as outlined in the following paragraphs:

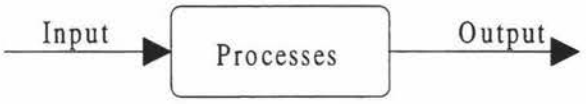


Figure 2.1- System Top Level

2.1- The Input:

- A stack of maximum 20 images. A stack is defined as a series of 2D images represented as a 3D structure. The third dimension identifies the serial location of a 2D image in the structure. Each image is an 8-bit grey scale image with resolution of 768×512. Two consecutive images have a minimum vertical overlap of 50 pixels (10%) and a maximum of 300 pixels (60%). There is a horizontal shift in both left and right direction up to a maximum of 50 pixels in either direction. A detailed account of the characteristics of the input images to the system is discussed in the next section.

2.2- The Processes:

- Finding vertical overlaps and horizontal shifts between consecutive images;
- Removing overlaps and shifting images for alignment and then stitching images together to get a single composite high resolution image;
- Indicating the validity of the results. The system has to be fail safe. If the input images are significantly distorted or there is no real match between frames the process has to be able to recognise that and prompt the user in an appropriate way.

2.3- The Output:

- A single composite high resolution image. The resolution of this final image depends on the overlaps and shifts of individual images that were used to construct the high resolution image. Smaller overlaps and shifts produce a higher resolution image output;
- A log file containing the following information:

a) Vertical overlaps and horizontal shifts information to be used by measurement software for calibration.

b) Validation of the results.

3- INPUT IMAGES

The input images to the system are primarily snap shots of sections of trees captured and digitised by a frame grabber card. They are originally recorded on a video tape and are taken in forest conditions. In the following paragraphs some important characteristics of the input images are discussed.

3.1- Light Variation

The lighting conditions in the forest vary greatly depending on the density of the forest, the time of the day and weather conditions. For a tree that is 40 meters high, the light level at the bottom of the tree where the camera is looking at the floor of the forest could be a few hundred times less than the top of the tree where the camera is essentially looking into the sky. The images at the bottom of the tree have relatively low contrast, but in a dense forest, due to the diffused nature of lighting at the floor of the forest show much detail. This is not the case at the boundaries of the forest. Here the camera may look into the light outside of the forest which is far more intense than the light level inside the forest. This results in a backlit situation where the captured image is of high contrast but much of the detail disappears. The images captured at the top of the tree are worse than those captured at the boundaries of the forest and the backlit situation is more severe at the top of the tree.

A full stack of images is taken over a time span of approximately 20 seconds. Sometimes the light level changes within this time due to moving clouds etc. However this variation is less significant than the change of light level due to the height at which different sections of tree being imaged. The image capture system is designed to deal with the change in light level at each section by allowing the camera to automatically adjust the aperture and integration time combination. This improves the quality of images. However, the camera can not compensate for the backlighting condition.

Figure 3.1 shows a typical input image to the system from the bottom and the top of a tree.

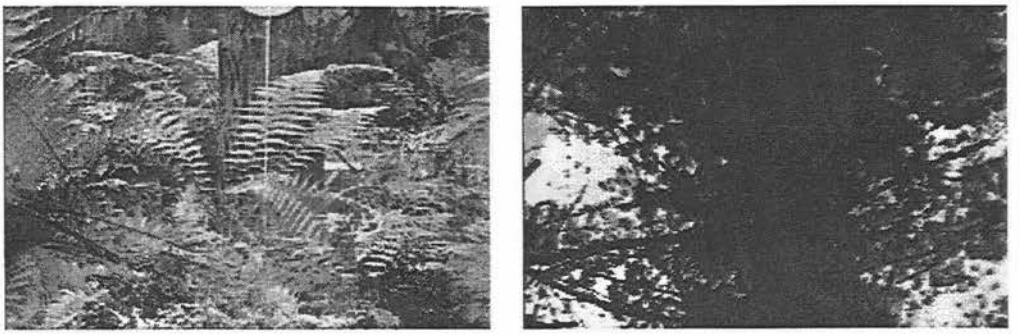


Figure 3.1- Typical input image from bottom and top of a tree

3.2- Video Timing

Another important characteristic of input images is the distortion caused by inconsistency of the video timing signals and the sampling process of the analog video from the tape. To understand the problem, a brief discussion of the timing format of the analog video signal and the sampling process performed by the frame grabber is beneficial.

The analog video signal produced by replaying the tape is a raster image in PAL format. When this signal is displayed on a monitor, a raster image is produced by scanning an electron beam across and down the face of the CRT in precisely positioned horizontal lines. The beam travels from left to right, and from top to bottom. The beam starts at the top left corner of the display and traces the first horizontal line by scanning across to the upper right corner. When it reaches the upper right corner of the screen, the beam skips downward and is repositioned at the left of the screen. The beam traces another line and this process is repeated until the beam reaches the last line at the bottom of the screen. The beam is then reset to the top of the screen and this process is repeated.

The period of time when the beam is tracing a horizontal line is called the *Active Video* portion of the signal. The period of time when it is reset from the right end of one line to the left edge of the next line is called *Horizontal Blanking Period.* The period of time when the beam is reset from the end of the last line at the bottom of the screen to the beginning of the first line at the top of the screen is called *Vertical Blanking Period.*

The action of beam is controlled by the horizontal and vertical sync pulses. These pulses are recorded on the tape along with the Active Video and define the timing and the duration of periods when the beam is repositioned.

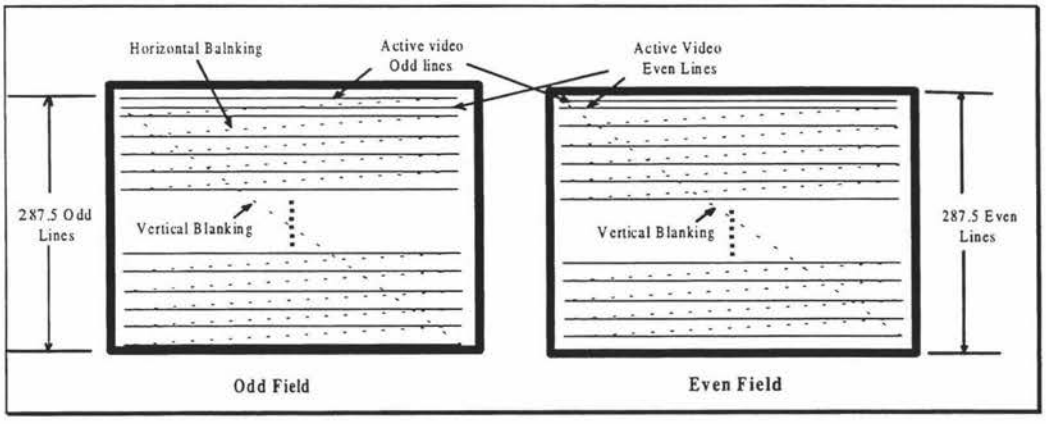


Figure 3.2.a- One Video Frame

A *frame time* refers to the time it takes to trace all the horizontal lines including all blanking intervals. The PAL signal contains 25 full *frames* each second. PAL *frames* are *interlaced* format signals. This means that each frame is divided into 2 separate *fields* of video information typically called *even* and *odd* fields (See Figure 3.2.a).

The *even field* contains all the even numbered lines in a frame and the *odd field* contains all the odd numbered lines. Interlacing involves displaying one field at a time. At the start of each frame, even lines are scanned one at a time starting with line 0 during the first field time which takes 1/50 of a second. When the last even line is scanned the beam is positioned to the beginning of the first odd line (line 1). Then during the second field time all the odd lines are scanned in between the even lines, thus the term interlacing. This interlacing cuts down on the visually perceptible flickering of an image changing every 1/25 of a second.

If the object and the camera are moving relative to each other then due to the interlaced format of the video the image in the even and odd fields will be misaligned. The amount of misalignment is directly proportional to the relative speed of the movement between the camera and the object. To avoid this misalignment problem the scanning platform stops at each section of the tree while the camera is recording frames from that section.

Each PAL *field* consist of 287.5 horizontal line periods which contain active video followed by a 25 horizontal line time *vertical blanking* interval. This interval contains, among other things a series of *vertical sync pulses*. So each PAL *frame* consist of 287.5 + 25 + 287.5 + 25 = 625 horizontal line times[2]. One horizontal line of video signal as observed at the output of a video camera is shown in Figure 3.2.

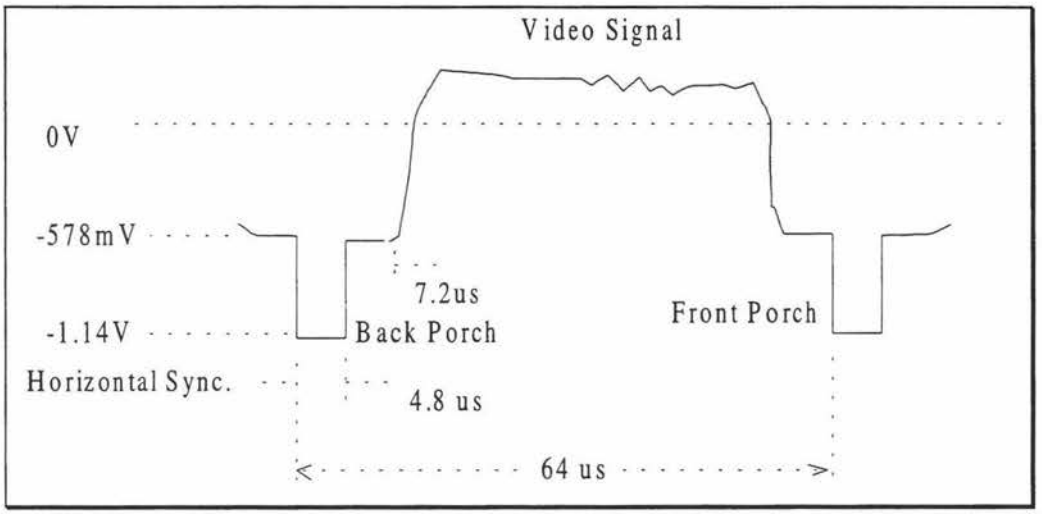


Figure 3.2- One horizontal line of video signal

The nominal horizontal line time is 64µs. However video signals may contain a lot of noise, making the determination of the sync edges unreliable. Particularly when the analog video signal is recorded on a magnetic tape and reproduced using a VCR, the noise on the video signal and variation in the horizontal line time is increased. For VCR, instantaneous line to line variations are up to ± 100ns. Line variations between the beginning and end of a *field* are up to ±5µs. When VCRs are in special modes such as fast forward or still picture, the time between horizontal sync signals may vary by up to ±20% from nominal [3].

Most modern VCRs perform head switching at the field boundaries, usually somewhere between the end of active video and the start of vertical sync. When head switching occurs, one video signal (for field n) is replaced by another video signal (for field n+1) which has an unknown phase offset from the first video signal. There maybe upto a ±1/2 line variation in vertical timing each field. As a result, longer than normal horizontal and vertical syncs may be generated. [3]

Apart from the noise in the video signals generated by the VCR, since all the timing information is record on a thin tape, any physical stretch to the tape will

increase the length of the tape and therefore the period of sync pulses. Any speed variation of the motor spinning the reading head of the VCR and speed variation in the motor driving the tape mechanism will also cause a variation of the period of the sync pulses.

The sampling process performed by the frame grabber card is synchronised to horizontal and vertical sync pulses present in the video signal. Any error in the sync pulse periods in the analog video signal can introduce errors in the digitised video.

3.3- Overlapping Lines

The number of overlapping lines between consecutive images is dependant on the lens magnification (zoom), angle of movement between consecutive frames and the distance of the camera from the tree. To obtain a uniform resolution for all the images taken by the system regardless of the distance of the tree and the camera, the magnification of the lens is adjusted according to distance so that the field of view at the bottom of the tree includes about 2 meters by 1.5 meters. This provides a uniform resolution for all images since each frame consists of 768 x 512 pixels which are distributed uniformly over the field of view providing a spatial resolution of about:

2000 mm / 768 pixels = 2.6 mm / pixel.

Since the angle of movement between frames is fixed at 3.6°,the number of overlapping lines between consecutive images is directly proportional to the distance between the tree and the camera. Figure 3.3 shows this relationship diagrammatically. Table 3.1 presents some typical distances and the associated number of overlapping lines.

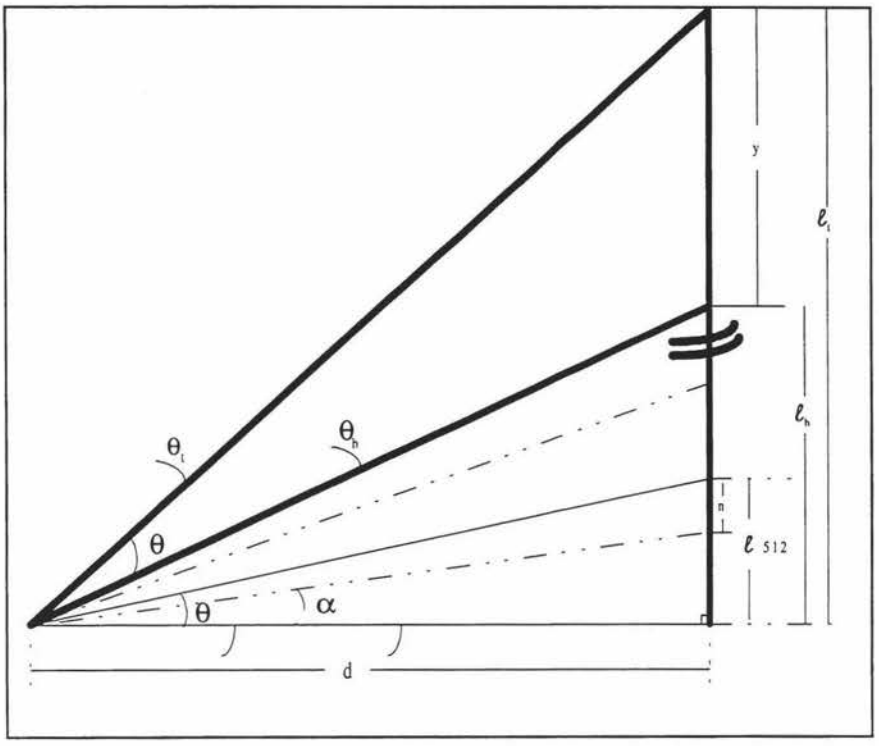


Figure 3.3- Distance and overlapping lines

As shown in figure 3.3:

$$\frac{Tan\theta}{Tan\alpha} = \frac{512}{512-n} \qquad \text{Equation 3.1}$$

For small angles of θ, α equation 3.1 approximates to:

$$\frac{\theta}{\alpha} = \frac{512}{512-n}$$

$$\Rightarrow n = \frac{\theta - \alpha}{\theta} \times 512 \qquad \text{Equation 3.2}$$

Table 3.1 is obtained using relationships in equation 3.2.

d (distance in meters)	θ (Vertical field of view in degrees)	n (No of overlapping lines in pixels)	α (angle of movement between frames)
10	**8.5°**	**295**	**3.6°**
15	**5.7°**	**188**	**3.6°**
20	**4.3°**	**83**	**3.6°**

Table 3.1- Overlapping lines variation due to varying distance

3.4- Image Resolution

Note that the field of view θ has been adjusted with the distance to keep the resolution of images constant. However, this image resolution is only constant in the first frame.

As the camera scans up the stem of the tree, the image resolution progressively decreases. This degradation of resolution is also a function of the distance between the camera and the tree. When there is a short distance between the camera and the tree, the resolution reduces at a higher rate than when the camera is further away from the tree. The perspective distortion is worse when the camera is close to the tree. Also note that the number of overlapping lines is a function of α, the angular movement of the scanning platform between consecutive frames. If the distance between the camera and the base of the tree is known, this angular movement can be adjusted based on the distance such that the number of overlapping lines remains the same for all the images regardless of the distance between the camera and the base of the tree. In the current design of the scanning platform this angle is fixed. In the following paragraphs the interdependency of horizontal and vertical resolution, distance of the camera and the base of the tree, and the height of the tree in the frame is further explored. As shown in figure 3.3:

$$\theta_t = \mathrm{Tan}^{-1}\left(\frac{\ell_t}{d}\right)$$

$$\theta_b = \theta_t - \theta$$

$$\ell_b = \mathrm{Tan}\theta_b \times d \qquad \textit{Equation } 3.3$$

$$y = \ell_t - \ell_b$$

$$V_R = \frac{y}{512} \qquad \text{Vertival Resolution}$$

The horizontal field of view is adjusted so that about 1.5m of the tree is inside a frame at the bottom of the tree. Therefore the angle of view in the horizontal direction is:

$$\beta = 2 \times Tan^{-1}\left(\frac{1}{d}\right) \qquad \textit{Equation } 3.4$$

The distance between the camera and the tree at the height of $\ell_b + (\ell_t - \ell_b)/2$ is:

$$d_h = \sqrt{(\ell b + (\ell t - \ell b)/2)^2 + d^2} \qquad \textit{Equation } 3.5$$

Field of the view of the camera in the horizontal direction covers:

$$x = Tan\beta \times d_h \qquad \textit{Equation } 3.6$$

Table 3.2 is calculated based on the relationships in Equations 3.3 through 3.6 and relates resolutions at different heights and different distances of the camera from the base of the tree:

Distance (m)	Height (m)	Theta (degrees)	Gamma (degrees)	Vertical Res(mm/pixel)	Horizontal Res(mm/pixel)
10	1	8.54	11.32	2.9	5.1
15	1	5.71	7.60	2.9	5.0
20	1	4.29	5.71	2.9	5.0
10	15	8.54	11.32	7.8	8.3
15	15	5.71	7.60	5.3	6.8
20	15	4.29	5.71	4.3	6.1
10	30	8.54	11.32	20.2	13.5
15	30	5.71	7.60	12.2	10.3
20	30	4.29	5.71	8.6	8.6

Table 3.2- Vertical and horizontal resolutions and field of views at different distances of camera from the base of the tree and at different heights up the tree

Figure 3.4 shows vertical resolution at different heights up the stem of the tree. It is assumed that the distance between the camera and the base of the tree is 15m. As can be seen from the graph, the slope of the line increases as the height increases. This relates to the increased perspective distortion observed high up the tree.

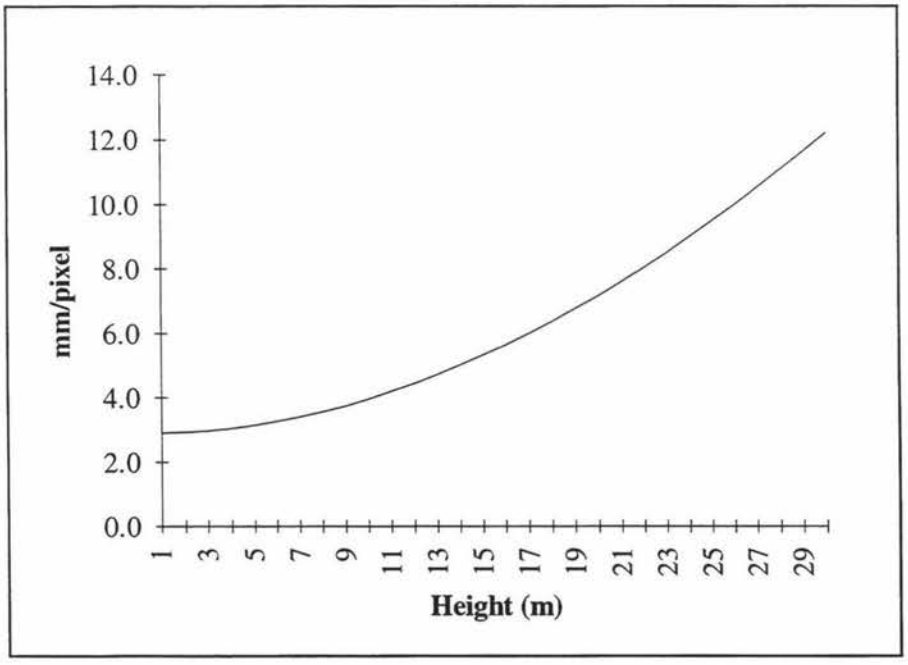


Figure 3.4- Vertical resolution at different heights when camera is positioned 15m from the base of the tree

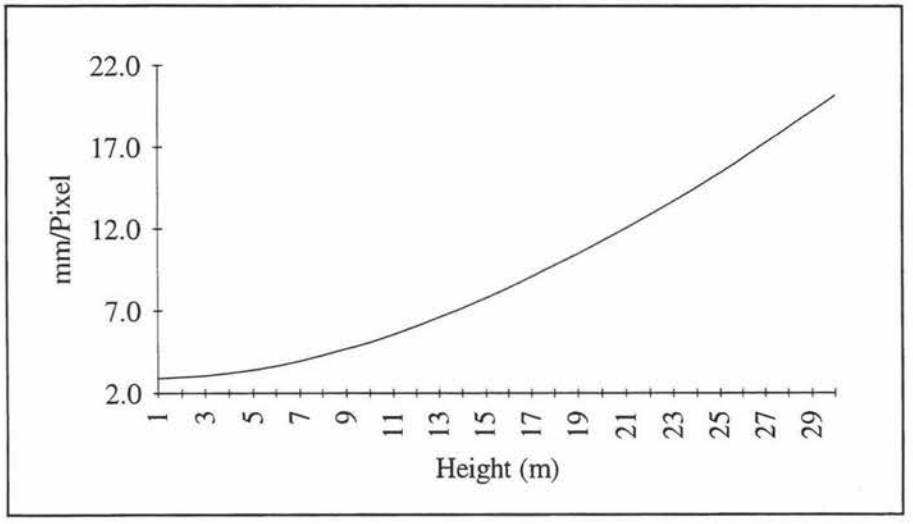


Figure 3.5- Vertical resolution at different heights when camera is positioned 10m from the base of the tree

Figure 3.5 shows vertical resolution at different heights up the stem of the tree when the camera is positioned 10m from the base of the tree. Comparing Figure 3.4 and Figure 3.5 shows a bigger slope in the resolution degradation line when the camera is closer to the base of the tree.

Figure 3.6 shows horizontal resolution at different heights up the stem of the tree where the camera is positioned 15m from the base of the tree. Comparing Figure 3.4 and Figure 3.6 shows that although the horizontal resolution of the image at the base of the tree is lower than the vertical resolution, at the top of the tree it is higher.

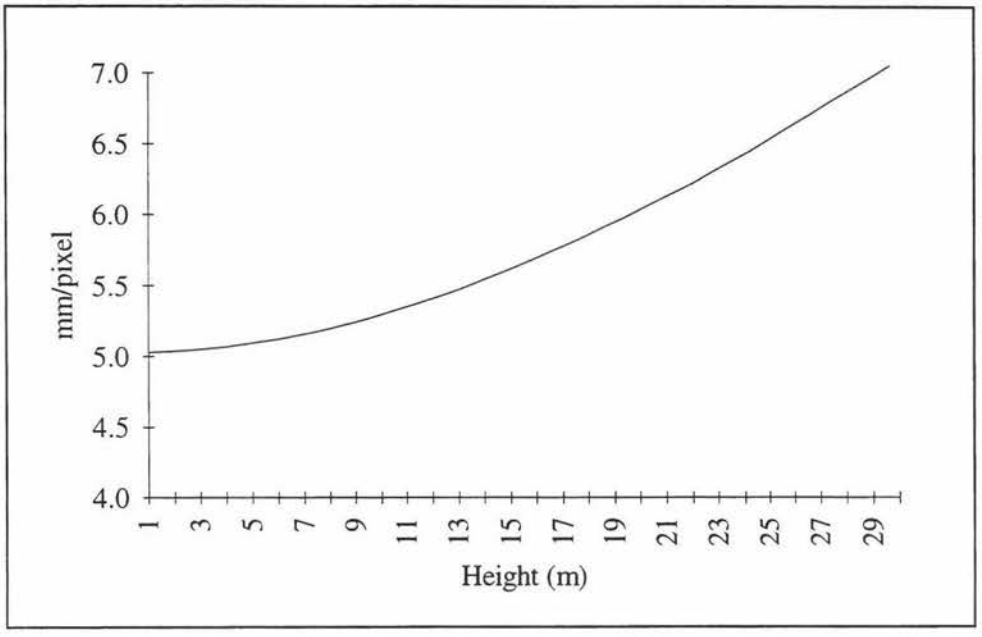


Figure 3.6- Horizontal resolution at different heights when camera is positioned 15m from the base of the tree

3.5- Perspective Distortion

This higher rate of resolution degradation in the vertical direction is the cause of the perspective distortion in the images. As the camera scans up the tree, the resolution both in the horizontal and the vertical direction decreases for two reasons:

- The distance between the camera and the section of the tree being imaged increases while the magnification is constant;
- The angle of view of the camera and the section of the tree being imaged changes.

This gives rise to the perspective distortion and also the different rates of degradation of the resolution in the horizontal and vertical directions. This effect is shown in Figure 3.7.

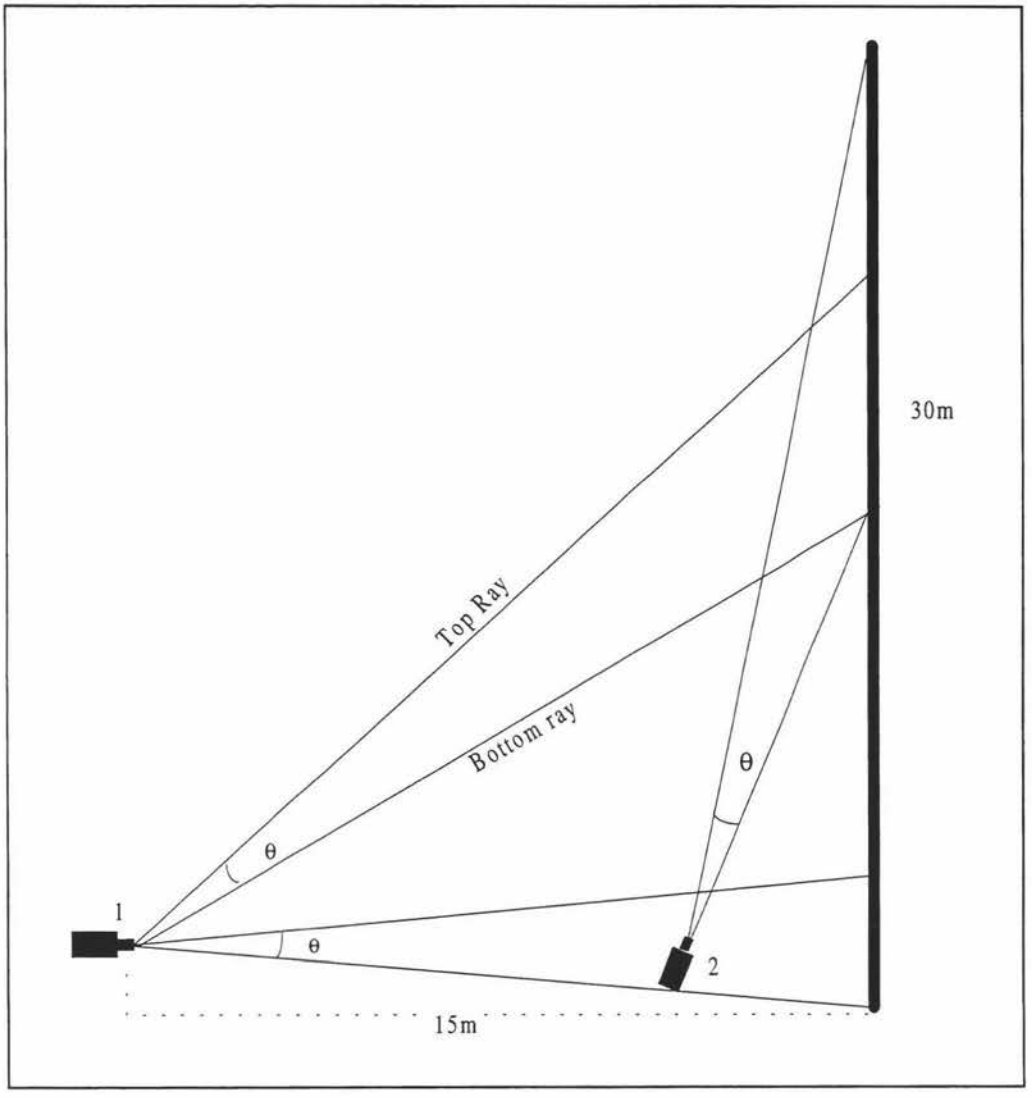


Figure 3.7- Different camera positions and angles

Figure 3.7 shows the vertical field of view of the camera at different distances from the tree as different heights of the tree are imaged. The magnification angle θ is the same in all situations. Note that in this position the top ray has to travel a longer distance to get to the camera than the bottom ray. This difference causes the perspective distortion. This effect becomes more severe as the angle of the field of view of the camera and the tree increases as shown in position 2 of the camera. Figures 3.8(a) and 3.8(b) shows images taken from positions 1 and 2 respectively.

Comparison of images at positions 1 and 2 shows that there is more perspective distortion when the camera is positioned closer to the subject and therefore the angle of the field of view is larger to provide the same physical dimensions of the subject in the view. These images show the top of a 30m high building

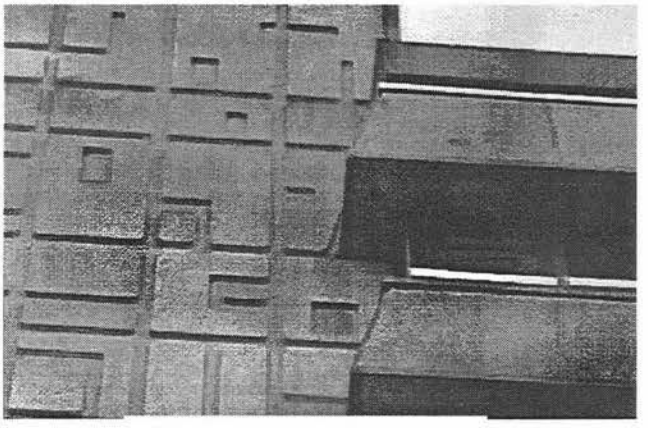


Figure 3.8(a)- Position 1

Figure 3.9 shows a frame at the base and one at the height of 30m up the stem of the tree. At the base of the tree, the field of the view of the camera includes 2m in the horizontal direction and 1.5m in the vertical direction. At 30m up the stem of the tree, the field of view of the camera includes 4.12m in the horizontal direction, a degradation in resolution of 4.12 / 2 = 2.06 times. However in the vertical direction, the field of view includes 6.25m which results in degradation in resolution of 6.25 / 1.5 = 4.2 times. This is about twice the degradation of resolution in the horizontal direction.

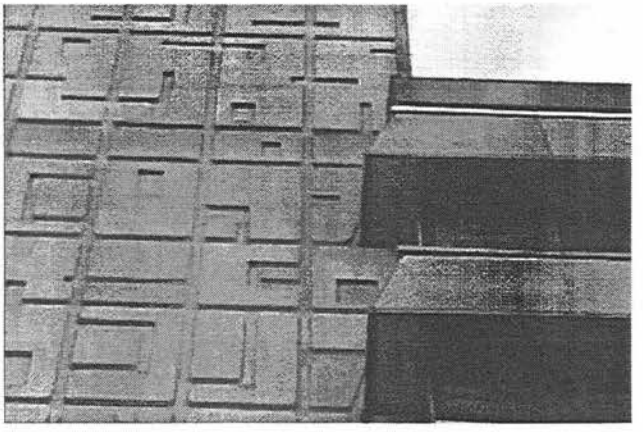


Figure 3.8(b)- Position 2

The degradation in the horizontal direction is due to the increased distance between the camera and the section of the tree being imaged since this distance is about twice the distance between the camera and the base of the tree. The degradation in resolution in the vertical direction is due to both the increased distance and the increased field of view of the camera.

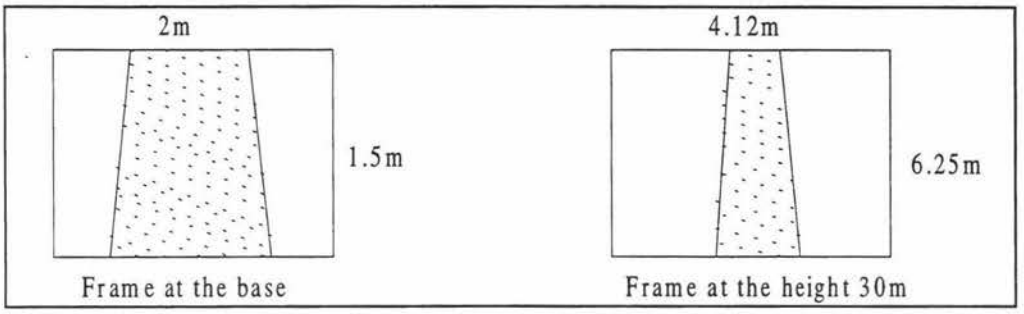


Figure 3.9- Frames at different heights, camera 15m from the base of the tree

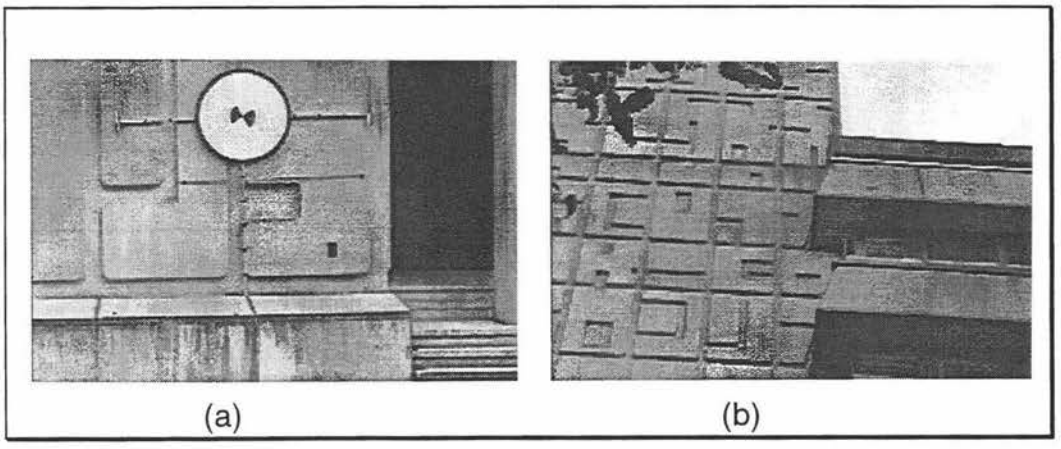


Figure 3.10- An image at the base of a building (a) and at the height of 30m (b)

Figure 3.10 shows an image of the base of a building and another at the height of 30 meters high up the same building. The horizontal distance between the camera and the building is 15.7m. Note the obvious perspective distortion in Figure 3.10.

Figure 3.11 shows two typical consecutive images input to the system.

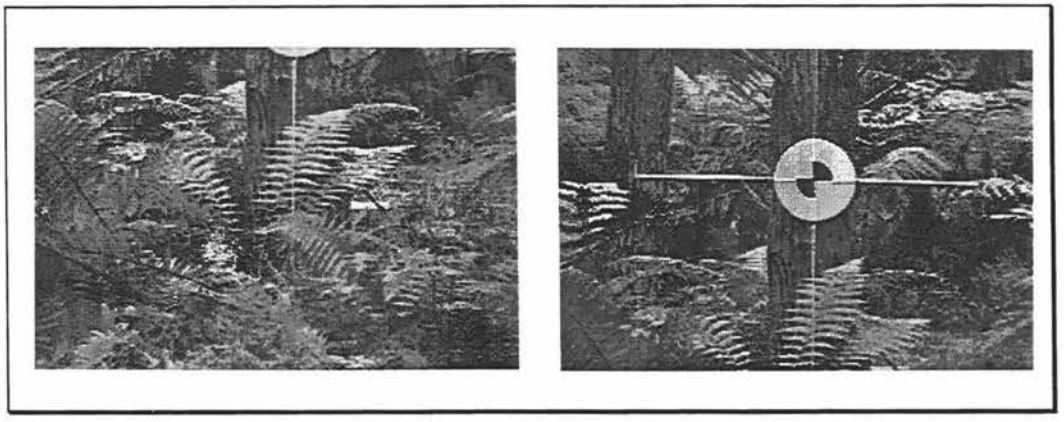


Figure 3.11- Typical consecutive input images

4-TOOLS AND BENCHMARKS

4.1- The Development Environment

The software package used as the development tool is NIH-Image [1]. NIH-Image is a freeware image processing package running under the Macintosh operating system and written by Wayne Rasband at the National Institute of Health, USA. NIH-Image is written in Pascal and its source code is available. It also support a Pascal like macro programming language which is an interpreted code and is suitable for many high level repetitive function calls that are computational extensive. Macro language is not suitable for low level pixel based operations due to the fact that it is interpreted code and is very slow. For customised and low level basic operations on individual pixels in an image, one can modify and recompile the source code to achieve a much higher speed.

The compiler used was Code Warior version 6 produced by Metrawerks Inc.

In the implementation of the first phase of this project, NIH-Image was selected as the development environment due to its availability and the flexibility that it provides for customised image processing tasks. In the first phase NIH-Image was used in conjunction with a frame grabber to acquire images from a video-tape and storing them in a stack This stack is the input to the work carried out in the phase 2 of the project.

The selection of NIH-Image in phase 1 proved to be helpful in the implementation of phase 2 of the project since the flexibility provided by both the macro programming language and the availability of the source code for modifications were necessary for this phase.

4.2- Benchmark Images

To test the system, a series of benchmark images are needed with known vertical overlaps and horizontal shifts. A manual method was devised to find the overlap and shift by a human operator using the computer. The procedure to find the match between consecutive images in a stack of 20 images is shown diagrammatically in Figure 4.1 and described in the following paragraphs:

1. Copy image I(n) to the clipboard;
2. Paste it on top of image I(n-1)
3. Perform bitwise XOR of the overlapping pixels. The value of the resultant pixel is an indication of the difference between the values of overlapping pixels. Pixels of the same value produce a zero value pixel. The bitwise XOR function is a very fast operation and can be performed much faster than subtraction operation which one might generally use to observe the difference between two images;
4. Move image I(n) on top of image I(n-1) and repeat the XOR operation and observe the intensity of the resulting image;

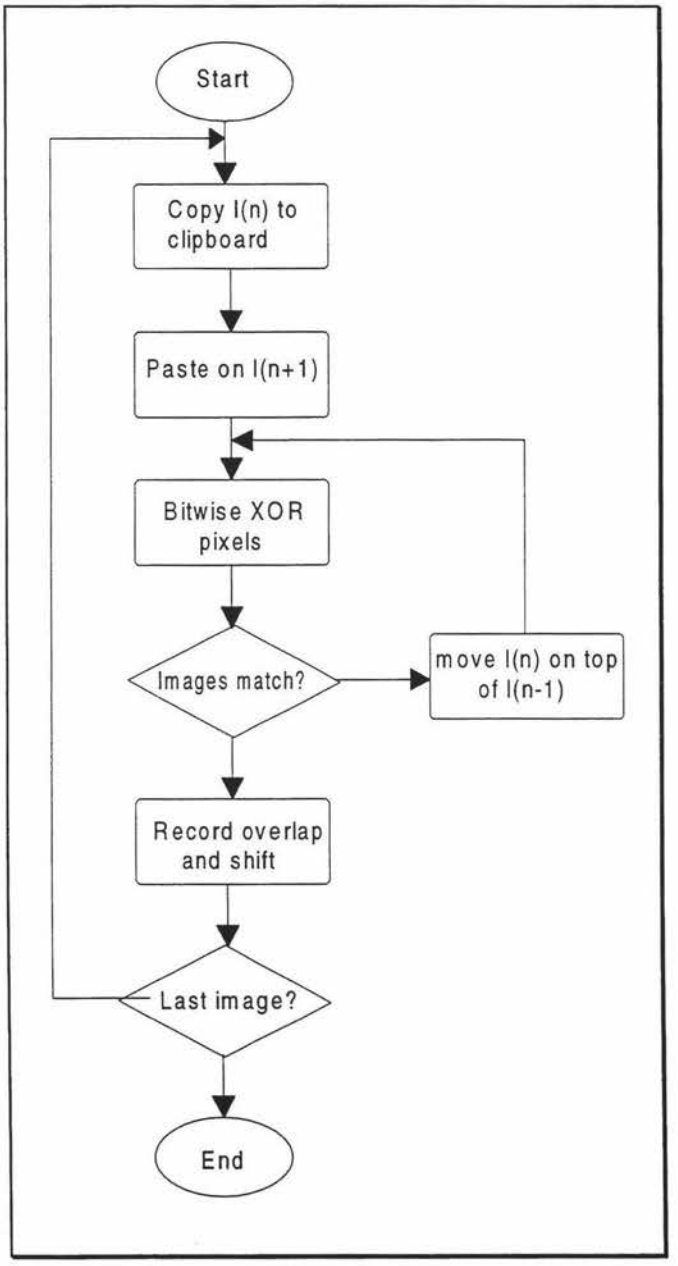


Figure 4.1- Manual Matching Flow Diagram

5. Repeat step 4 and look for a resulting image of maximum intensity. In NIH-Image maximum intensity corresponds to minimum pixel value;

6. Read the location of the edge of the area of the image with the maximum intensity. The Y-value of the edge pixel shows the vertical overlap. A vertical shift is always downwards and represented by a positive number. The X-value shows the horizontal shift. A horizontal shift of image I(n) to the right hand side on image I(n-1) is represented by a positive number and a left shift by a negative number. To assign a direction to the X-value, if the edge pixel is at the left hand side of the maximum intensity region then the X-value represents a right shift and therefore is assigned a positive sign. If the edge pixel is at the right hand side of the maximum intensity region then the X-value represents a left shift and therefore is assigned a negative sign;
7. Repeat steps 1 to 6 for images I(n) and I(n+1) and then for images I(n+1) and I(n+2) and so on until matches between all consecutive images are found.

Using NIH-Image, this method takes about one minute to find a match between two images. NIH-Image facilitates implementation of this method by providing a paste function combined with XOR function (and a few other functions such as logical bitwise OR, AND, and mathematical functions such as +,–,×,/ etc.). This paste function is particularly fast with bitwise logical operation which is a single cycle operation in the instruction set of the microprocessor. The bitwise XOR function is calculated and the result is displayed as one moves an image on top of another.

Using this method the vertical overlap and the horizontal shift can be found by the human operator with a certainty of mostly ±1 pixel and occasionally up to ±3 pixels. This uncertainty in the result is a consequence of noise and perspective distortions present in images and not inherent in the method employed. If the consecutive images were identical, apart from an XY translation, then the vertical overlaps and horizontal shifts could be found exactly. This was proven by producing a target image from a small selection of a larger image. Since there was an exact match for the target image in the larger image, one and only one vertical overlap and the horizontal shift were found .

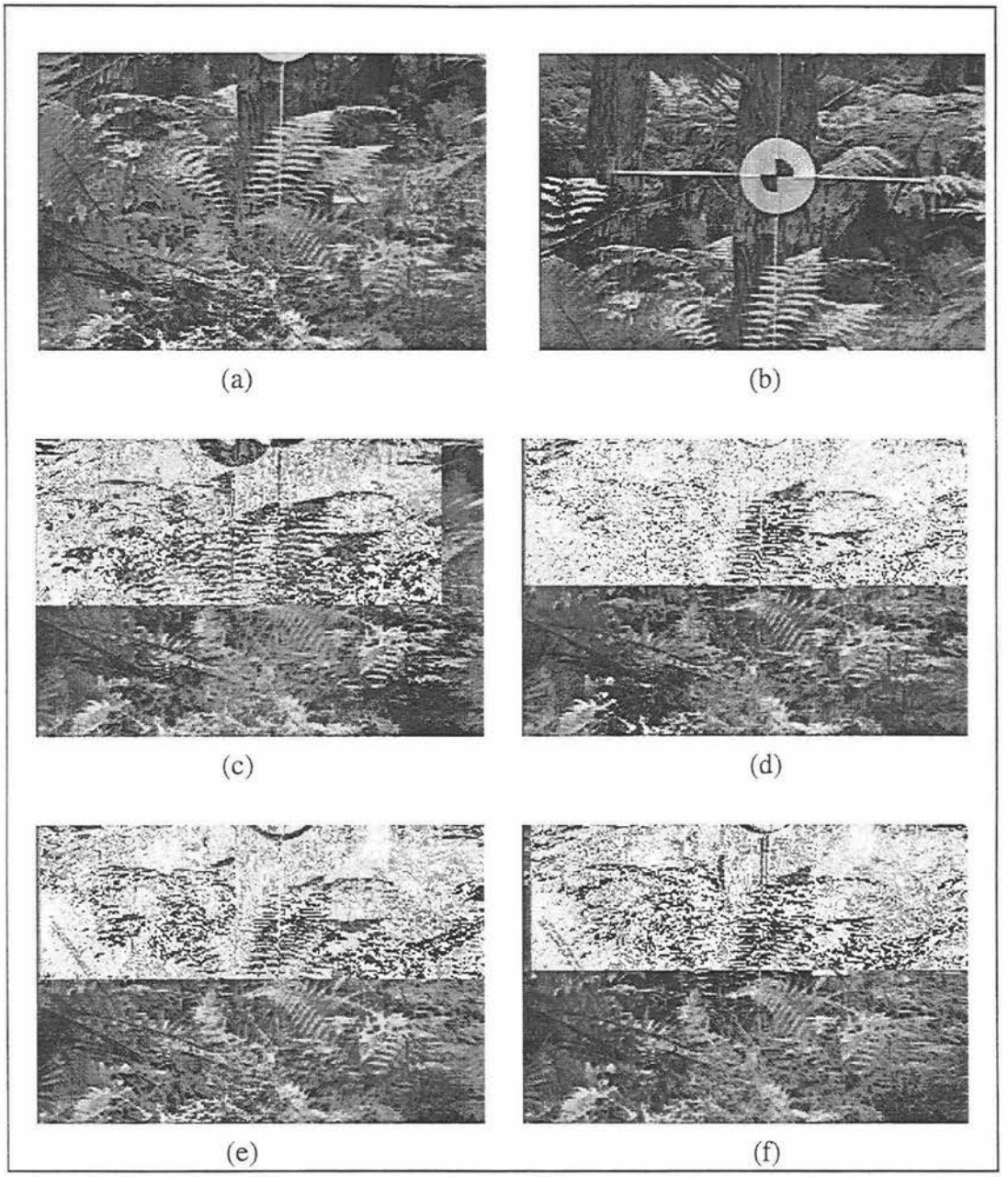


Figure 4.2- Sample images in manual matching

An example of implementation of this method is shown in figures 4.2(a) through (f). Figure 4.2(a) and (b) show the images to be matched. Figure 4.2(c) shows an unmatched resultant image. Figure 4.2(d) shows the resultant image where a match has been found. Figure 4.2(e) shows the resultant image where the images are offset by 2 pixels downwards from the actual match position. Figure 4.2(f) shows the resultant image where the images the images are offset by 2 pixels to the right from the actual match position.

5- MATCHING ALGORITHMS

5.1- Overview

If one is assigned the job of matching two photographs which include some common features, and if the exact displacement of one photograph on the other is required then a logical and practical method would be to put one image on top of the other and move it across and up and down until the features match. Then measure the amount of movement that was required to get to the match position.

The methods developed here for the computer to perform the matching are essentially the same as a human might do, except that it is automatic. However, it is not necessarily faster or more precise. It only takes a few seconds for the human vision system to perform the same task with a high degree of accuracy and reliability. There is no doubt that the ample processing power of human vision system often defeats an advanced computer systems.

Matching algorithms developed here fall into two broad categories: Spatial Domain Processing and Frequency Domain Processing.

An electrical waveform is often represented as a plot of magnitude of the voltage against time. Likewise a digital 2D image is an arrangement of magnitudes of pixel intensities in 2D SPACE. A photograph is an example of spatial domain representation. Two of the methods developed here for matching, Strip Matching and Block Matching, manipulate images in this form of representation and are examples of Spatial Domain Processing.

The same electrical signal or digital image can be represented by a plot of magnitude of the signal or pixel intensities at different frequencies. A final method developed here uses cross correlation in the frequency domain and manipulates images in this form of representation. This method is an example of Frequency Domain Processing.

Following is a brief account of the developed methods:

- **Strip Matching**- To find the match between image 1 and 2, a strip of pixels is taken from the bottom of image 1 and is compared with the strip of the same size from the top of the image 2. The absolute sum of differences between individual pixels in the strip is then calculated. This strip is then

moved around to the left and right to find the horizontal shift and down to find the vertical overlap (see Figure 5.0). The same operation is repeated for each strip location. The location with the minimum absolute sum of differences is the best match.

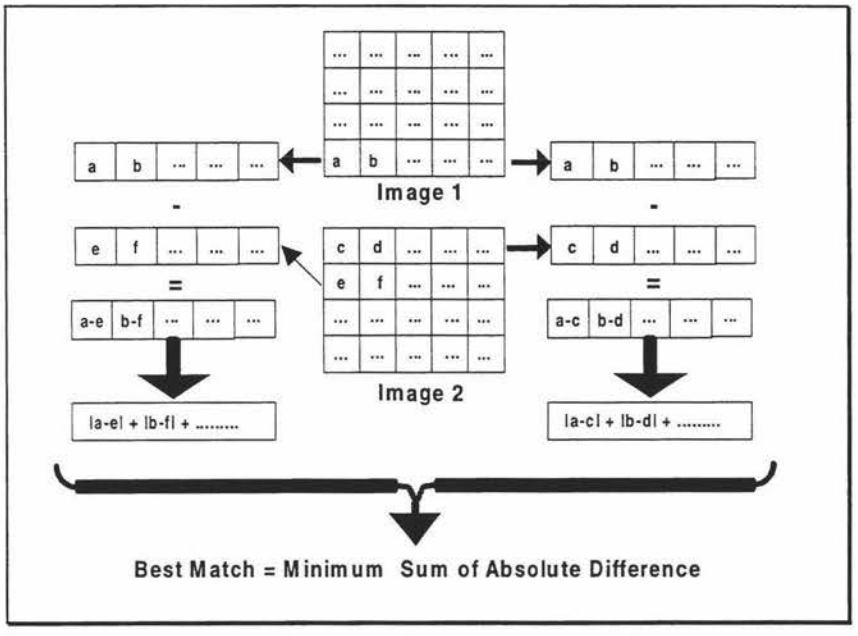


Figure 5.0- Strip Matching procedure

- **Block Matching-** With the strip matching method it is possible that the strip does not contain any significant feature to be matched to in the other image. Block matching tries to solve this problem by first searching on image 1 to find a small block of pixels which contain the most significant features. Then this block is used for matching. The matching algorithm is the same as strip matching method.

- **Cross Correlation (FHT)-** With both the strip and block matching methods a very small fraction of the actual overlap is taken from image 1 for finding a match in image 2. It is reasonable to assume that if a bigger portion of the actual overlap is taken to find a match for, there is a higher probability of finding the correct match. In cross-correlation method a bigger block is used for matching. However the computation time is directly proportional to the number of pixels used for matching. To reduce the required computation time for matching, this algorithm is implemented in the frequency domain.

The details of the above methods is discussed in the following sections.

5.2- Strip Matching

Strip matching algorithm is implemented using macro programming capability of NIH-Image as well as modification and recompiling of the source code. The higher level part of the algorithm concerned with manipulating images within the stack and creation of the single composite high resolution image is implemented in the macro language. This part is common to all different algorithms. That part of the algorithm concerned with pixel based operations to find the actual vertical overlap and horizontal shift is implemented by modifying the source code and recompilation of the program. This was done because the compiled code is faster by at least two orders of magnitude in dealing with iterative pixels manipulations within the image. On the other hand using macros for higher level image manipulation is just as fast as the compiled code since a macro function actually makes a call to a compiled code. The only overhead in using the macro is a function call and associated parameter passing and the saving of the state of the program at the time of the function call. This overhead is a small fraction of the actual processing time of the compiled called function. Macro programming is more convenient and flexible for image processing algorithm development.

5.2.1- Implementation in Macro Language

Figure 5.1 shows the flow diagram of the part of the strip matching algorithm implemented in macros. This program is modular and most parts are also used for implementation of the block matching and the frequency domain processing. Some of the important steps in the algorithm is explained in the following paragraphs. Macro code can be found in Appendix B.

- **Break stack to images-** A series of overlapping images are captured to give a complete coverage of the tree. Such an assembly is called a stack. The size of a stack is the size of one of the images times the number of images in the stack. A typical stack produced by the system contains 20 images each 768× 512 pixels (one byte). Therefore the size of a typical stack is about 7.86MBytes.

$$768 \times 512 = 393 KBytes$$

$$20 \times 393 = 7.86 MBytes$$

Memory requirements and usage by NIH-Image is as following:

- NIH-image application about 1M;
- Total size of the open image(s) and text file(s);

- Undo and Clipboard buffers which are equal and are set by the user. The size of the buffers must be larger than the largest single open image so that all the undo, copy and paste operations can be performed properly;

To process a typical stack in NIH-Image, 7.86MBytes of memory will be allocated to the stack. A blank image of about 6Mbytes is created to accommodate the produced high resolution image. Undo and Clipboard buffers need to be set to at least 6Mbyte to accommodate for the largest single image which is the blank image. Note that although the size of the stack is 7.68Mbytes, it is divided to 20 smaller individual images and the undo and clipboard operations are performed on individual images rather than the whole stack. Operations such as open, close and save are performed on the whole stack which do not require clipboard or undo buffers. Therefore the minimum required memory for the NIH-Image to process a typical stack is about 27MBytes.

1 *MByte*	*Application*
7.86 *MBytes*	*Stack*
6 *MBytes*	*High* Re *s* Im *age*
6 *Mbytes*	*UndoBuffer*
6 *MBytes*	*Clipboard Buffer*
26.86 *MBytes*	*Total*

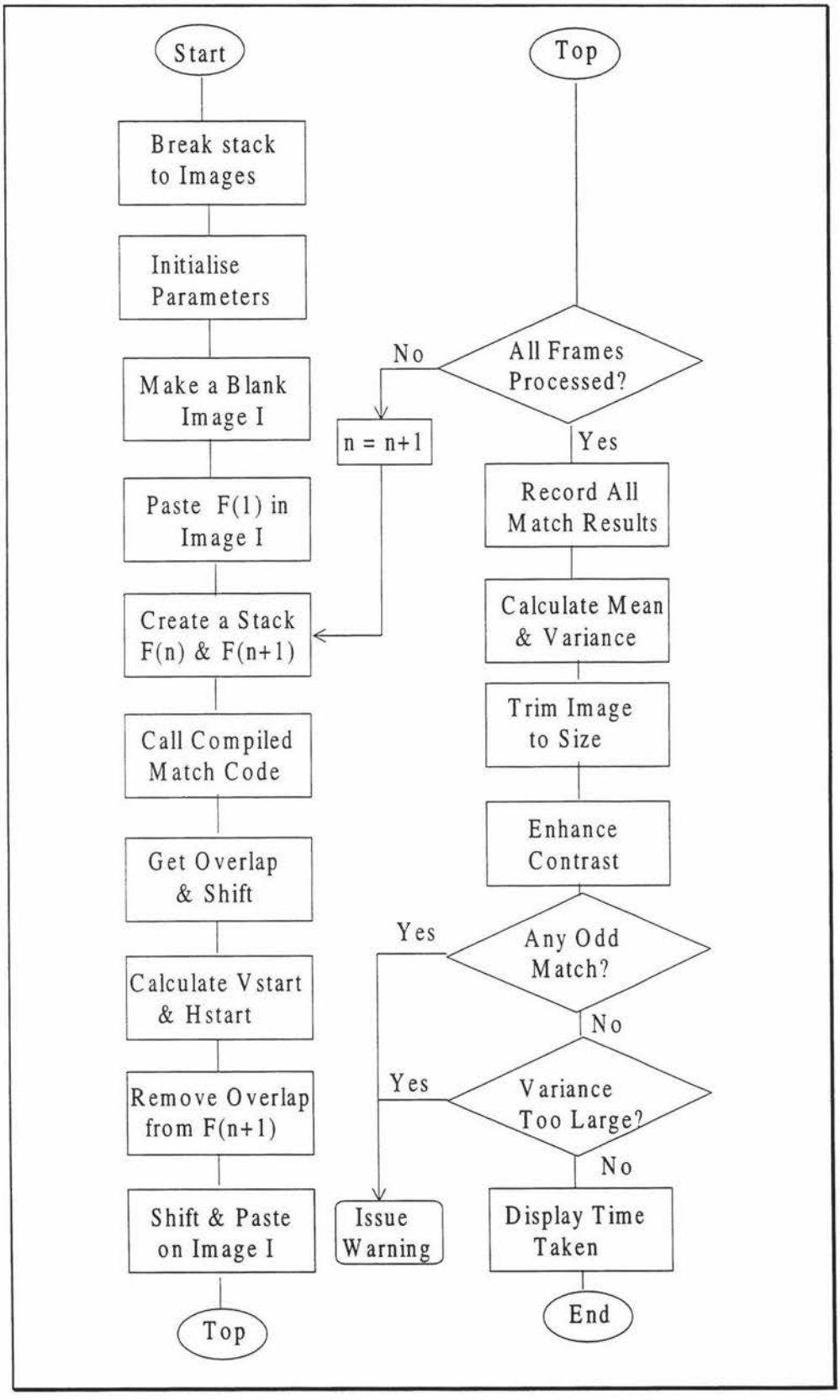


Figure 5.1- The strip matching algorithm implemented in the macro programming language

A few MBytes are also required for the computer Operating System. Therefore the minimum memory requirement for the system is 32MBytes. This algorithm was further developed to use 24 MBytes memory by reducing the memory requirement of the stack to less than 1MByte. This was achieved by breaking the stack into its individual images and saving each image on disk with a filename which reflects the position of the image in the series in the stack. Only two consecutive images are opened at a time since only two images are required to find the shift and overlap.

- **Initialise Parameters-** There are a few parameters that are used in the strip matching algorithm which are shown in Figure 5.2 and explained below:

  1. Strip- Indicates the number of lines of one image that are taken for finding a match in the other image;
  2. Vstart- This parameter is used as the starting point in the vertical direction to find the overlap;
  3. Vend- Indicates the end of the search for finding overlap in the vertical direction;
  4. Vrange- Indicates the range of search in the vertical direction for finding the overlap. Vrange is the difference between Vstart and Vend. These latter parameters are calculated based on the most recent vertical overlap and the Vrange.
  5. HrangeStart- Indicates the range of horizontal search used for the first match where there is no knowledge of the horizontal shift available yet.
  6. Hrange- Indicates the range of search in the horizontal direction for finding the shift. This value is used after the shift between the first two images is known.

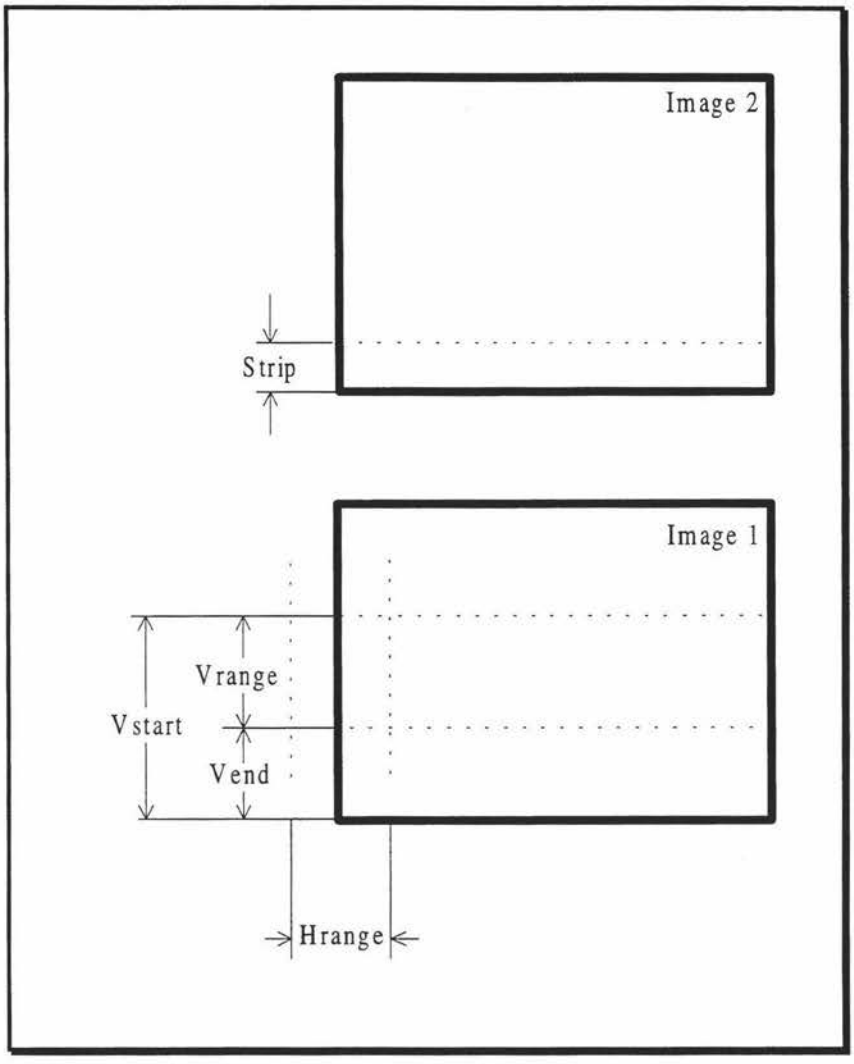


Figure 5.2- Some of the parameters used in strip matching

- **Make a blank image-** A single high resolution image is formed from non-overlapping parts of the lower resolution images. This high resolution image is first set up as a blank image and as the lower resolution images are processed and the overlap between them is determined, the non-overlapping parts of each image are pasted in an appropriate place into the blank image. In this way the high resolution image is gradually formed as the lower resolution images are processed. Since the size of the high resolution image depends on the number of overlapping lines and shifts in the consecutive frames and these values are not known at the beginning of the processing, the blank image has to be as large as possible so that all processed frames can be included in it. An assumption is made that the vertical overlap is a minimum of 10% (51 lines) and the maximum horizontal shift is Hrange. Therefore the size of the blank image is:

$$BlankHeight = n \times Height / 1.11$$

$$BlankWidth = Width + Hrange,$$

where:

width is the width of a single frame in the stack;
Height is the height of a single frame in the stack;
BlankHeight is the height of the blank image;
BlankWidth is the width of the blank image;
n is the number of frames in the stack.

- **Paste F(1) onto the Blank Image I-** To make up the high resolution image, the non-overlapping parts of a lower resolution image (frame) is pasted onto the blank image. The first frame in the stack is pasted in its entirety and the overlap is taken off the next frame in the series.

- **Make a stack from F(n) and F(n+1)-** As it was mentioned earlier to reduce the memory requirements of the algorithm, a stack is broken into individual images. To process images, a small stack comprising of only two images is constructed. The algorithm only requires two consecutive images to find the vertical overlap and the horizontal shift between them.

- **Call compiled Match code-** This is a routine which, given a stack of two images, will find the vertical overlap and the horizontal shift between them. This routine will be described in detail in the next section.

- **Get Overlap and Shift-** The result of the Match code is vertical overlap and horizontal shift. These values are passed into the macro code through the built in user arrays User1 and User2.

- **Calculate Vstart and Hstart-** To reduce the time of calculation the range of the search is limited in vertical direction to start from Vstart and end at Vend and in the horizontal direction from Hstart to Hend. These parameters are calculated based on the known characteristics of the system and the result of the match in the previous images in the stack. Vrange is a parameter which reflects one of the characteristics of the system and is an indication of how much the overlap between two images can vary. In the vertical direction,

once the vertical overlap is obtained by processing two images in the stack, Vstart and Vend are calculated:

$$Vstart = Voverlap - Vrange/2$$

$$Vend = Vstart + Vrange$$

Hstart and Hend are calculated based on the same procedure.

- **Remove overlap from frame F(n+1)-** Once the vertical overlap is obtained, this overlap will be removed from the bottom of the frame producing an image that can be stitched to the partially formed high resolution image.

- **Shift and paste on Image I-** The produced image in the above section will now be shifted to the right or left according to the horizontal shift obtained in the Match code and pasted on top of the partially formed high resolution image I.

- **Record match results-** If all the frames have been processed and the vertical overlaps and the horizontal shifts are obtained, all the results will be written into two pre-defined arrays in the NIH-Image namely User1 and User2. These results can be displayed, saved as part of the high resolution image or as a separate file for further processing of the high resolution image.

- **Calculate mean and variance-** One of the requirements of the system is that it has to be fail safe. For the purpose of the matching algorithm fail safe means that if an unexpected match is found, a warning has to be issued to indicate the possible problem. This is done through calculation and consideration of the mean and variance of individual match points between images and will be discussed in detail in the latter sections.

- **Trim image to size-** As it was mentioned in the previous section, at the beginning of the processing a blank image is created to accommodate the high resolution image. This blank image is intentionally made larger than the final high resolution image to ensure there is enough space for the largest possible high resolution image. Once all the frames are processed and there are no more frames to be pasted onto the initially blank image, this image needs to be trimmed to the actual size of the high resolution image. The height of the actual high resolution image is calculated as following:

$$FinalHeight = n \times Height - \sum_{i=2}^{n} Overlap(i)$$

where:

Height is the height of a single frame in the stack;
FinalHeight is the height of the high resolution image;
n is the number of frames in the stack;
Overlap(i) is the vertical overlap between frames i and i-1

- **Enhance contrast-** This is an image processing function provided by the NIH-Image software. Enhancement of the contrast is achieved by convolution of the following kernel with the high resolution image.

-1	-1	-1
-1	12	-1
-1	-1	-1

Since most of the images are taken in the low light conditions typical of forest situations, the contrast of the images is not very good and the appearance of the final high resolution image can benefit from contrast enhancement operation.

- **Warning Message-** A warning message is issued if the results of the matching process does not meet the confidence requirements.

- **Display of processing time-** At the end of the processing of the stack, total time taken is displayed. This is useful for comparing the processing time of different algorithms as well as quantifying the effect of changing processing parameters such as Strip size, Vrange and Hrange etc.

Most parts of the algorithm described in the previous paragraphs are common to all matching algorithms. These are mainly high level image manipulations or function calls to built-in image processing functions in the NIH-Image software. This modularity of the software design simplified the trial and implementation of different matching algorithms. The actual matching algorithms are mainly implemented as a function call to a compiled code.

Figure 5.3 shows the top level representation of the part of the strip matching algorithm implemented in the Pascal programming language.

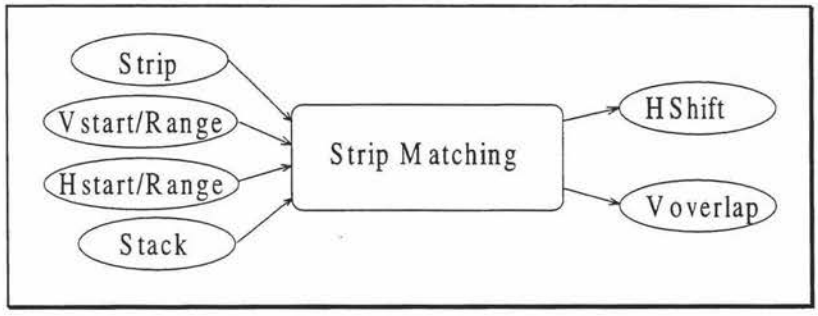


Figure 5.3- Top level diagram of Strip matching

5.2.2- Implementation in Pascal Language

Figure 5.4- shows the flow diagram of the part of the strip matching algorithm implemented in Pascal source code. For the source code of this algorithm refer to Appendix C. In the following sections the steps in the algorithm are explained.

- **Check for stack-** The input image to the matching algorithm has to be a stack. The algorithm first checks for a stack and if the input is anything but a stack, an error message will be displayed and the algorithm ends.

- **Get strip from image 1-** A strip is defined as part of an image with the same width as the image and a height specified by a parameter (Strip) passed to the algorithm by the calling macro as the strip height . The goal of the algorithm is to find a match for this strip in the other image.

- **Position strip on image 2-** A strip of the same size need to be taken from image 2 at a specific position for comparison with the strip taken from image 1. This is analogous to positioning the strip taken from image 1 on top of image 2 at a specified location and then processing the overlapping pixels. The starting coordinates of the positioning is specified by the macro program as Hstart and Vstart and the ending positions as Hend and Vend.

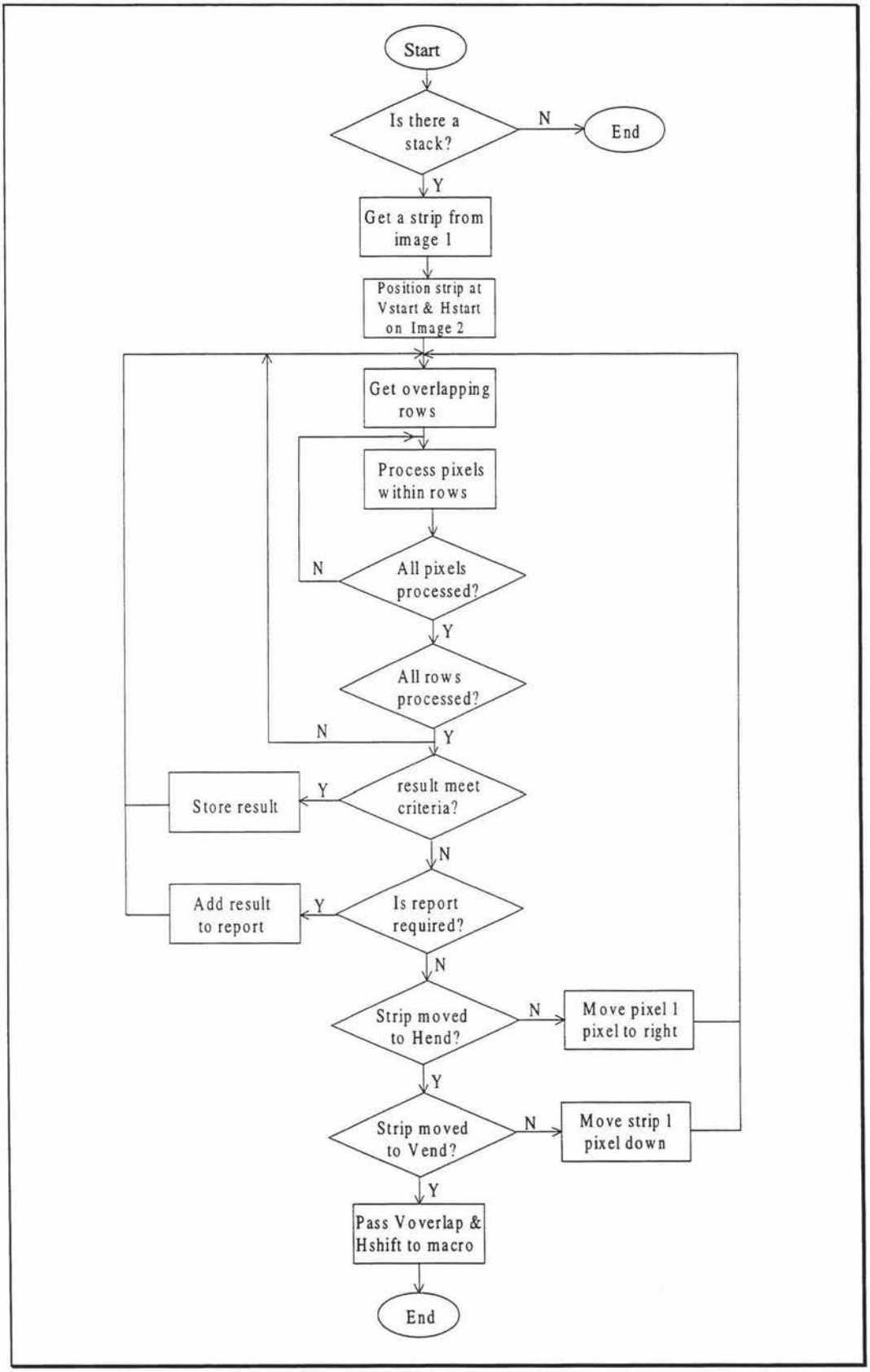


Figure 5.4- Flow diagram of the part of the strip matching algorithm implemented in Pascal source code

5.2.3- Processing Time

The time taken by the matching algorithm is directly related to the number of pixels involved in the process and the computational complexity of the process. Therefore to reduce the total computation time one or both of these contributing factors need to be reduced. In the following section each of these factors will be discussed in some detail.

5.2.3.1- Pixel Count

Number of pixels involved in the operation is:

$$No.\ of\ Pixels = Strip \times Width \times Hrange \times Vrange$$

Where: Hrange=Hend - Hstart

Vrange =Vend - Vstart

For example, on a normal image of 768 pixels wide and a strip size of 5 lines, Hrange of 20 pixels and Vrange of 100 lines, the total number of pixels that need to be processed is 7.68 million.

To reduce the total number of the pixels any of the above factors may be reduced to some degree without greatly effecting the accuracy of the result. A suitable size for the Strip and the Width is somewhat dependant on the features of the image where the size of the Hrange and Vrange depend on the accuracy and repeatability of the movement of the scanning platform at the time of recording the images on the tape.

If the orientation of most features in the image are horizontal then having a larger strip size will help in detecting the features in the other image. This is because a large vertical selection of such an image will include many of the features which may be matched with the features in the other image. If, however, most of the features are aligned in the vertical direction, a wider selection of the image will be beneficial since a wider selection has a better chance of including the distinctive vertical features which may be located in the other image.

Orientation of the features in a typical forest image are mostly vertical with some other features in random orientations. For this reason it was decided to use a small strip size and a large width which stretches across the image. Typically a

strip size of 5 lines and the full width of the image is used for tree images with satisfactory results.

As was mentioned above, the Hrange and Vrange values are dependent on the accuracy and repeatability of the scanning platform. Theoretically, once the vertical overlap and the horizontal shift between the first and second frame is found, the other frames must have the same overlaps and shifts. Hshift variation is a result of small movement of the camera on the scanning platform in the horizontal plane and it is normally small. Vertical overlap variations is related to differences in the angular movement of the scanning platform from one frame position to the next. This is mainly caused by the mechanical inaccuracy in the scanning platform motor and gearbox and camera platform combination. The effect this has on the image is dependant on the distance of the camera from the base of the tree. This is because regardless of the distance of the camera from the base of the tree, the camera is zoomed so that a frame contains 2 meters of the tree in the horizontal direction and 1.5 meters in the vertical direction. This has the consequence that the angle of the field of view is dependant on the distance. Each frame contains 512 lines in the vertical direction which corresponds to the vertical angle of the field of view. For example if the angle of the field of view is 7.2° (with the fixed angular movement of 3.6°, there will be about 50% overlap between consecutive frames) then the angle per each line of the image is:

$$7.2^{0}/_{512} = 0.014^{0}/_{Line}$$

A variation of angular movement of 0.014 results in one line difference in the overlapping lines in two consecutive images. Table 5.1 presents the equivalent angle for each line of the image at different distances of the camera from the base of the tree and the results are graphed in Figure 5.5.

Distance	Vertical Field of Vie	Degree/Line
10	8.54	0.0167
11	7.77	0.0152
12	7.13	0.0139
13	6.59	0.0129
14	6.12	0.0120
15	5.71	0.0112
16	5.36	0.0105
17	5.05	0.0099
18	4.77	0.0093
19	4.52	0.0088
20	4.29	0.0084

Table 5.1- Equivalent angle for each line at different distances

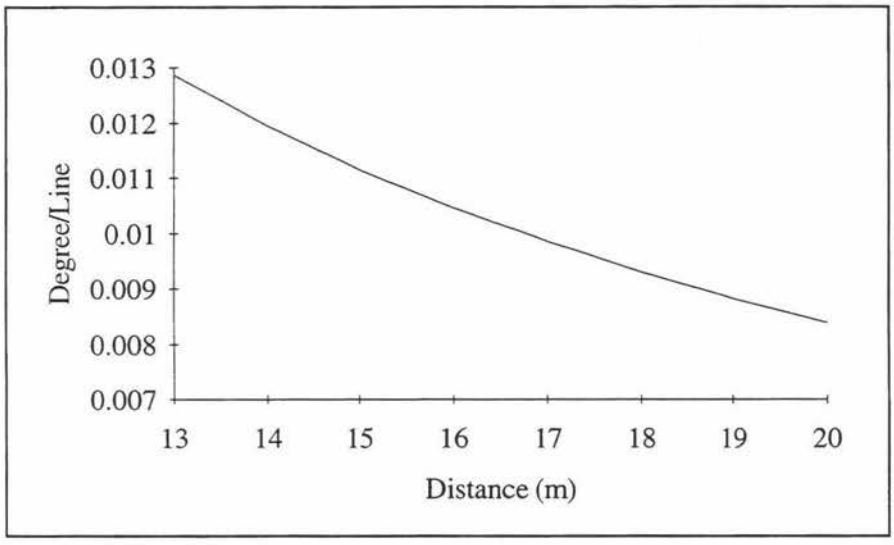


Figure 5.5- Equivalent angle for each line at different distances

As it can be seen from table 5.1 and Figure 5.5, the error in angular movement becomes more significant as the distance between the camera and the base of the tree increases.

Examining the images taken by the system, typical variation in the vertical overlap is about ±10 lines at the distance of 15 meters. This corresponds to a variation of angular movement of the scanning platform between consecutive frames of about:

$$0.0112 \text{ Degrees/Line} \times (\pm 10 \text{Lines}) = \pm 0.112 \text{ Degrees}$$

The maximum variation in the vertical overlap of ±18 lines was observed in a sample of 12 images. The Vrange variable in the algorithm must be larger than this maximum to ensure that all overlaps will be found. Vrange of 40 lines was chosen which is about 10% larger than the maximum variation in the vertical overlap between any two consecutive images.

The maximum variation in the horizontal shift of ±10 was observed in the same sample as above. The mean variation in the horizontal shift was less than 3 pixels. There were some horizontal shift values which were significantly different from others. This indicates a sudden movement of the camera on the scanning platform in the horizontal direction which perhaps has happened because of loose fixing of the camera on the tripod[1]. If this sudden movement is reliably eliminated, then a maximum horizontal shift variation of ±3 will be obtained. To achieve high reliability in detection the horizontal shifts ±10 pixels was used as the maximum variation and Hrange of 22 pixels which is over 10% higher than the maximum variation was chosen.

[1] Camera has a very thin and weak fixture for mounting it on the tripod. It does not lend itself to very tight fitting on the tripod. One fixture broke in field trials and had to be replaced.

5.2.3.2-Process Complexity

To find the match between two images, some operation must be performed on each pixel in the overlapping area between the strip taken from one image and the other image. Two equivalent operations were considered:

- **Minimum mismatch energy-** Mismatch energy for a strip u(m,n) and the selection of the image v(m,n) is defined as [4]:

$$\sigma^2(p,q) = \sum_m \sum_n [v(m,n) - u(m-p, n-q)]^2$$

This is the sum of the square of the differences between individual pixels in the strip and the underlaying pixels in the other image corresponding to a horizontal shift of p and a vertical shift of q. Where this quantity is minimum, there is the best match for the two images and p and q indicate the overlap and shift between the two images.

This operation requires a byte subtraction, multiplication and addition for each individual pixel. Using this process on a Power PC 7100 with a processor speed of 66MHz a match can be found between two images in approximately 4250 ms. To speed up the process, absolute value of the difference can be used rather than the square. This will reduce the processing time to 1653 ms; approximately 3 times faster.

- **Cross Correlation-** It can be shown that the above operation is equivalent to finding a maximum for the cross correlation [4]:

$$c_{vu}(p,q) = \sum_m \sum_n v(m,n) \times u(m-p, n-q)$$

This operation requires a byte multiplication and addition for each individual pixel. Using this process on the above computer a match can be found between two images in approximately 1700 ms.

Following is the best combination of factors which yields the fastest total processing time to find a match using strip matching algorithm without sacrificing reliability:

Strip =	5
Hrange =	22
Vrange =	40
Process =	Minimum Mismatch energy using absolute values

5.3- Block Matching

In the strip matching algorithm, a strip of pixels is taken from the bottom of one image and used for finding a match in the other image. However the bottom of one image may not contain many significant variations in the pixel values and therefore, it may not be possible to find a correct match for it in the other image. In the block matching algorithm, this problem is overcome by first finding a block of pixels in one image where there are significant variations in the pixel values. Then this block is used for finding a match in the other image. There are two processes in the algorithm: finding the block and then matching the block.

5.3.1- Find Block Process

The Find Block process and the inputs and outputs are shown in Figure 5.3.1.

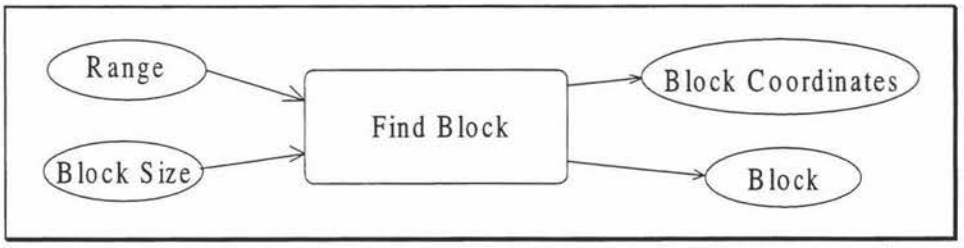


Figure 5.3.1- Find Block process

The inputs to the process are:

Range- This value specifies the vertical range of search. If the Range is one, only the bottom of the image is searched to find the block. However if the Range value is greater than one, then a number of blocks above the bottom of the image equal to the Range will also be searched.

Block Size- This is the size of the block to search for, size is typically set to less than 100x100.

The outputs from the process are:

Block Coordinates- This indicates the coordinates of the left, top pixel of the block.

Block- This is the actual pixel values of the area which was found within the range of the search to have the most significant variation in the pixel values and therefore the most features.

5.3.1.1-Flow Diagram

The flow diagram of the "Find Block Process" is shown in Figure 5.3.2. In the following paragraphs some key steps of the flow diagram are explained. Source code can be found in Appendix C.

Place ROI- A ROI of the size "Block Size" is selected at the bottom of the image such that its left top corner is positioned at the left corner of the image at a height equal to:

$$\left(Range-1\times n/2\right)+n$$

where n is the height of the block.

Move Down- In each loop of the process the ROI is moved down by half the height of the block and to a total number equal to the "Range". Moving down by half the height of the block gives 50% overlap between adjacent ROIs in the vertical direction. Any amount of overlap can be chosen. However, larger overlaps leads to longer processing time without much gain in the accuracy of the results since useful features to look for are likely to be greater than half of the height of the block.

Move Right- Once one ROI is processed it is moved to the right by half the width of the block. This process continues until the ROI is reached to the rightmost pixel of the image.

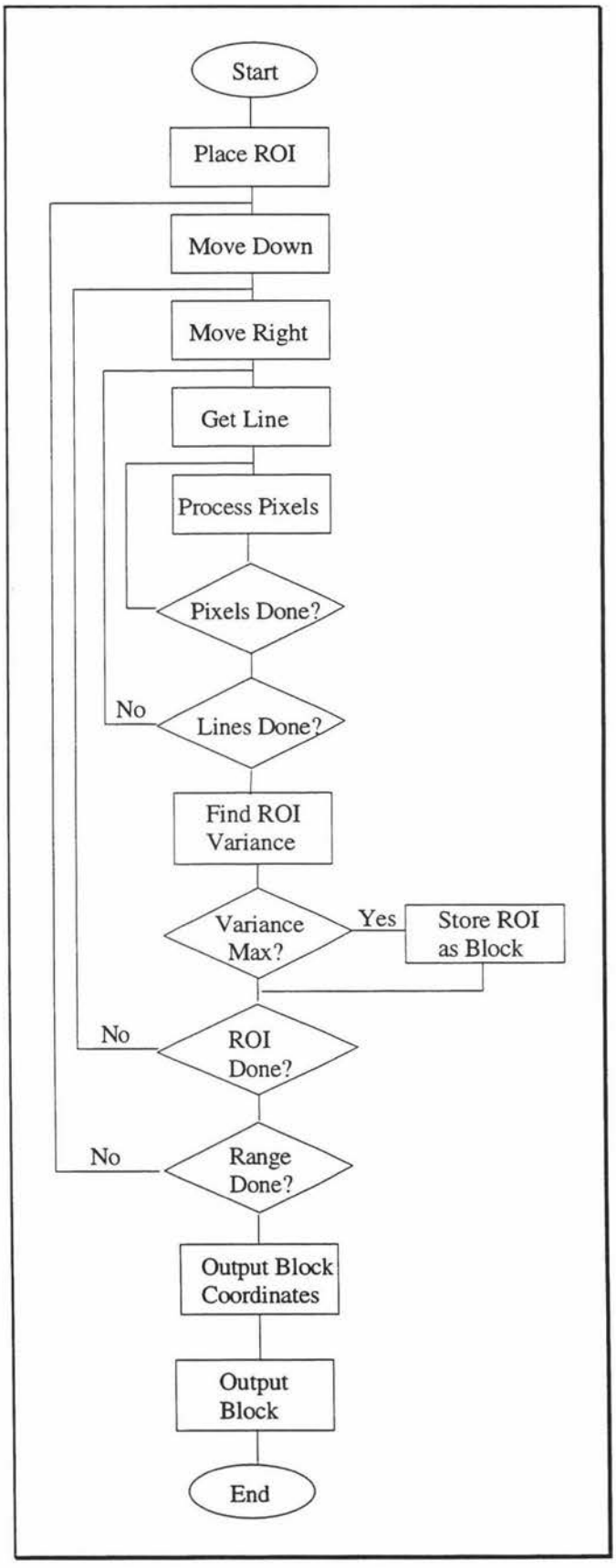


Figure 5.3.2- Flow diagram of the Find Block Process

Get Line- At each ROI position, one line at a time is taken from the ROI and pixels within it are processed until all the lines in the ROI have been processed.
Process Pixels- The selection of the block is based on the maximum variance in the pixel intensities within the ROI. The variance is defined as:

$$\delta^2 = \frac{1}{n}\sum(x-\bar{x})^2$$
$$= \frac{\sum(x^2)}{n} - (\bar{x})^2$$

where x is the pixel value and $\bar{x}$ is the mean value of all n pixels within the ROI. Using this equation, all the pixels should be first processed to find the $\bar{x}$. Then all the pixels should be processed again to find the δ^2.

Since $\bar{x}$ is equal to $\frac{\sum x}{n}$ therefore:

$$\delta^2 = \frac{\sum(x^2)}{n} - \left(\frac{\sum x}{n}\right)^2$$
$$= \frac{1}{n}\left(\sum(x^2) - \frac{(\sum x)^2}{n}\right)$$

Equation 5.3.1

Using equation 5.3.1 the calculation of the δ^2 only requires sum of x and x^2 which is calculated in only one pass and therefore will reduce the processing time.

Find Variance- Once the sum of x and x^2 of each block are obtained, the variance of the ROI is calculated using Equation 5.3.1. If this variance is larger than the maximum variance of the other ROIs previously calculated, then the coordinates of this ROI is stored as the current maximum variance.

Output- The ROI with the largest variance is output from the process as the "Block" and the coordinates of its left top corner pixel as the "Block Coordinates".

Figure 5.3.2 shows an image and the plot of the variance of a 20×20 block running across the bottom of the image.

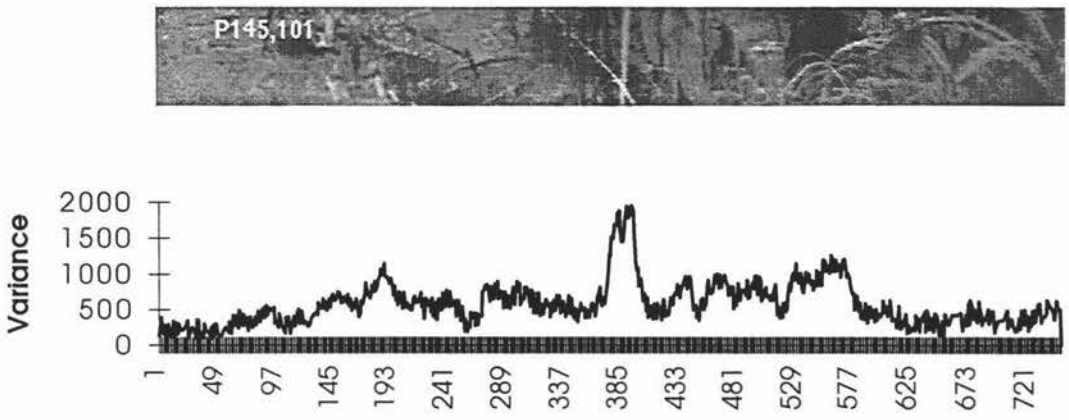


Figure 5.3.2- plot of variance of a 20×20 block running across the bottom of the image

5.3.1.2- Processing Time

The processing time to find the desired block is directly dependant on the two inputs to the process: The "Block Size" and the "Range". The larger these values are, the greater the chance of the block containing significant features. Suitable sizes depend on the characteristics of the image. If the image is "busy" with relatively large number of features which extend across the image, then there is a high probability that a small block size searched in a small Range will contain significant features. However, if the image is plain or the features are only local then a larger block size and search range is required. Recall that the basic reason that this algorithm was devised was that the strip matching algorithm did not have a high success rate on plain images.

Experiments on tree images show that to benefit from the use of this algorithm, block sizes larger than 30×30 must be used. This can be explained by way of an example. Suppose it takes one unit of time to find the result for a 30×30 block with the Range value equal to 1. If the Range is increased to 2 or 3 the processing time will be 2 or 3 units of time. If however the Range value remains the same but the block size is increased to 60×30 and 90×30 then the time will still increase to 2 and 3 units of time respectively but in this case a larger area of the image has been examined to find the block with the most significant features. A block size of 30×60 with the Range value of one will analyse the same area of the image as a block size of 30×30 and the range value of 3. In the latter case the processing time will be 1.5 times of the former because some areas have been processed twice due of the 50% overlap. Combination of small block size and a range value greater than 1 is useful where the image contains small isolated features in a large background.

Note that this process is only a part of the algorithm. Once a suitable block is identified, another process will use this block to find a match for it in the second image. This will work faster with smaller block sizes. Therefore it may be better to spend more time in this process with a larger Range value to get a smaller block size which will be processed faster in the next process. The next process and its execution time will be discussed in the next section. Table 5.3.1 summarises the processing time for various block sizes and Range values in the Find Block process. This algorithm was run on a Macintosh computer with a PowerPc 66MHz processor and 28 Meg of memory.

	Range			
	0	1	5	10
Block	Processing Time (sec)			
30x30	1.1	2.0	6.0	11.0
60x30	1.9	3.8	11.2	20.5
90x30	2.7	5.4	16.0	29.2

Table 5.3.1- Processing Time of various block sizes and Range values

Experimenting with tree images showed that a block size of 50x50 and a range of 3 gives satisfactory results.

5.3.2- Match Block Process

Once a block is found to have significant features, then a match for it should be found in the second image. This is done in the Match Block process. The block diagram of the Match Block process is shown in Figure 5.3.3.

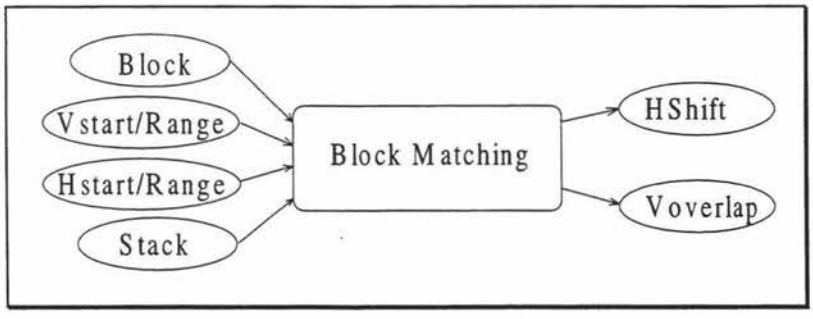


Figure 5.3.3- Top level diagram of Match Block process

The match block process is very similar to the strip matching process. In the strip matching process a block of pixels in one image is searched for in the second image. The main difference is that in the strip matching process this block is chosen blindly without initial processing, while in the block matching algorithm the block is chosen specifically because there is a high probability that it contains significant features.

Another difference is that the strip in the strip matching algorithm is thin in the vertical direction and long in the horizontal direction (typically 5×768) where the block used in the block matching algorithm is generally square (typically 50×50).

For details of the match block process refer to the section 5.2.2.

5.3.2.1- Processing Time

Processing time is a linear function of the inputs to the process: Vrange, Hrange and Block size. Increasing any of the inputs will increase the processing time. Since this process is very similar to the strip matching process, for details on the processing time refer to the section 5.2.3.

Table 5.3.2 summarises time taken by the match block process to find a match for the block given different inputs into the process.

	Vrange			
	40		100	
	Hrange			
	10	20	10	20
Block	**Processing Time (ms)**			
30x30	1.7	2.4	2.8	4.4
60x30	2.8	3.6	4.1	6.0
60x60	5.6	7.0	8.0	11.8

Table 5.3.2- Match Block processing time

Total processing time of the block matching algorithm is the sum of the processing time of its two processes: Find Block and Match Block.

The following is the best combination of factors which yields fastest total processing time to find a match using block matching algorithm without sacrificing reliability:

Block Size =60×30
Range =5
Hrange =22
Vrange =40

5.4- Frequency Domain Matching

5.4.1- Background

Time varying signals can be represented in the frequency domain. While the time representation of the signal shows the strength of the signal at different times, the frequency representation of the same signal shows the strength of the signal at different frequencies. Both representations contain useful information and one can be transformed to the other.

When a continuos time varying signal is sampled, it becomes discrete. Each sample point will show the amplitude of the signal at the time of sample. This sampling process results in many points which are separated in time by the sampling period. Likewise a digital image is made from individual pixels positioned at different locations in a 2 dimensional space. Each pixel can be regarded as a sample which is separated from the adjacent sample by a space. Therefore images are space varying signals. Since any kind of variation will have a corresponding frequency, images can be represented by the frequency of their pixel value variation.

Images are initially captured in the spatial domain. To perform an image processing operation in the frequency domain the image has to be first transformed into its frequency domain representation. Although there are many different types of image transformations that can be used, the most well known is the Fourier Transform developed in 1807 by Baron Jean-Baptiste-Joseph Fourier. Fourier Theory states that any continuous one dimensional periodic function f(x) can be represented exactly as a summation of an infinite series of sine and cosine terms of increasing frequency [5]. One important characteristic of this transformation is that an inverse transform can be performed on the series of the frequency terms to reconstruct the original function f(x). Images, however, are not continuos functions. They are a series of discrete numbers. Discrete Fourier Transform (DFT) applies to a finite count of samples in time or space domain and produces a series of numbers of the same count of increasing frequency terms. This algorithm has a very efficient and fast implementation on computers called Fast Fourier Transform (FFT) developed in 1965 by James Cooley and John Tukey[5]. There is also a more recent algorithm called Hartley Transform developed by Hartley [6] and its fast implementation for digital computers called Fast Hartley Transform (FHT) by Bracewell[7]. The Hartley transform distinguishes itself from its close cousin, the Fourier transform, by being real valued; it produces real output from real input. Even so, it provides the same phase and amplitude information about the data as the Fourier transform. The Fast Hartley transform (FHT) is twice as fast as a complex valued FFT, requiring virtually the same number of operations as the real

valued FFT algorithms. The Hartley transform is so symmetric that the forward and inverse transforms differ only by a multiplicative constant [8]. Some mathematical background on both FFT and FHT and also implementation of algorithms in NIH-Image software are included in Appendix D.

The pixels in an image have real (integer) values but the corresponding pixels in the resulting frequency domain image have complex values. These complex values provide an amplitude and a phase at each pixel location. A complex valued image can not be displayed in a meaningful way. Therefore often only the amplitude or the power spectrum (square of amplitude) is displayed. In a power spectrum image, the DC value which is a representation for the average pixel values in the image is displayed as the origin and higher frequencies are separated from the origin at a distance proportional to the frequency. The amplitude or the power at each location shows how much of that frequency is present in the original image and the different directions show the orientations in the original image(see Figure 5.4.1). It is important to note that although a frequency domain image is often displayed using only the amplitude information, the original image can not be reconstructed from this image alone and the phase information is required in addition. It can be shown that the phase information is even more important than the amplitude information in reconstruction of a transformed image into its original spatial representation[14].[1] However displaying the phase information will not convey visually useful information and is very hard to interpret.

To illustrate these points a few ideal images in the spatial domain and their power spectrum in the frequency domain are displayed in Figure 5.4.1.

[1] One could Fourier transform two VERY dissimilar images, separate out the amplitude & phase parts of the frequency spectrums and then reconstruct very good images from the PHASE spectra when these were recombined with the WRONG AMPLITUDE data!!

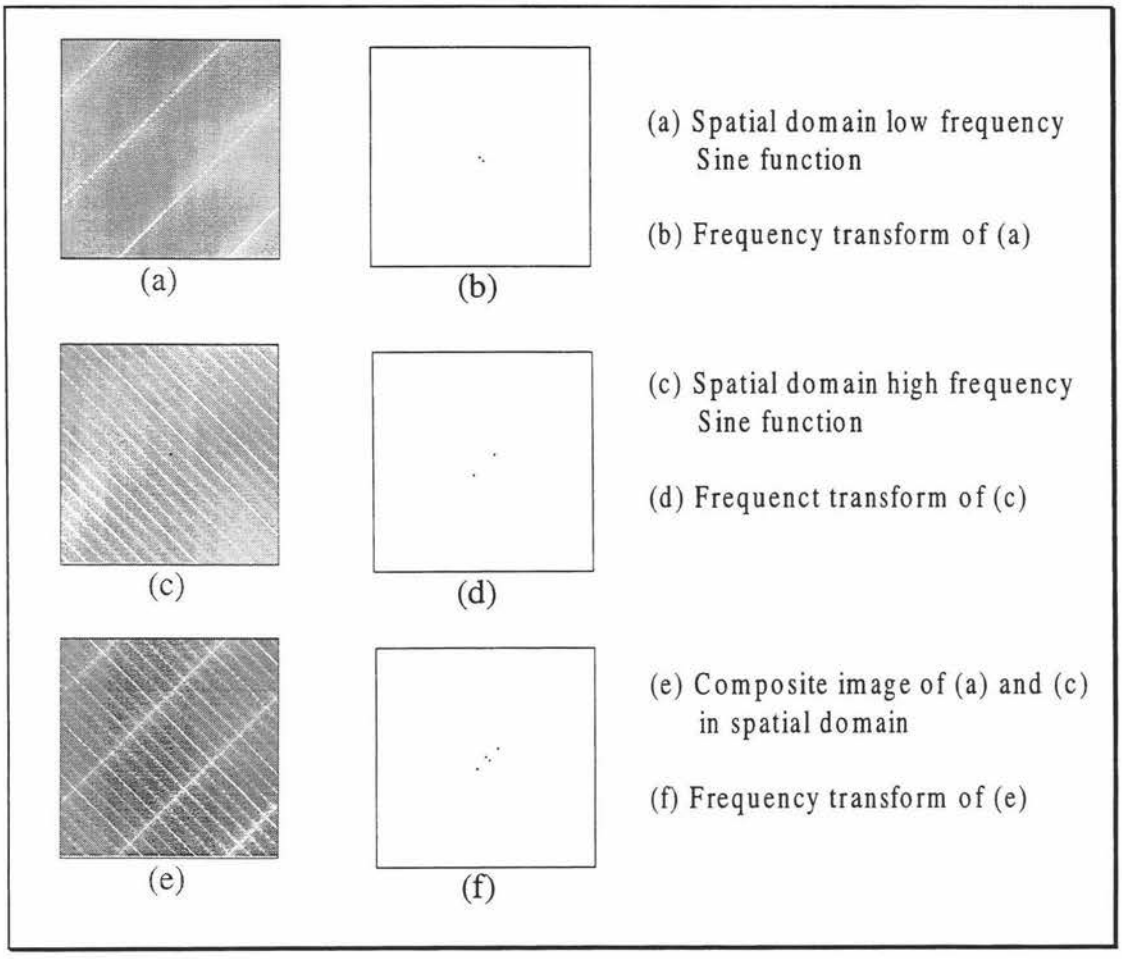


Figure 5.4.1- Spatial and frequency domain of some Sine function images

Figure 5.4.1 is a good example of transformation of a periodic signal. It can be seen that the pure sinusoidal variation in brightness of pixels in each spatial representation image, produces single and symmetric peaks in the frequency domain whose distance from the origin indicates the frequency of the sine wave and the direction of it shows the orientation . It also clearly shows that the transformation of an image which is formed by the superposition of a few images, results in the superposition of the frequency domain representation of those images. If any pair of the points in the frequency domain representation of the composite image is deleted and the resulting image is inversed transformed to the spatial domain, then the resulting image will not contain the sinusoidal patterns related to those pair of points which were deleted [9]. This is an important application of the frequency transform of images to remove a periodic noise superimposed on the entire image by deleting just a pair of points in the frequency domain image and performing an inverse transform to the spatial domain.

Another important application of frequency transformation of images is performing correlation which is most relevant to this paper. It can be shown that the correlation of

two images in the spatial domain is exactly equivalent to the pixel by pixel multiplication of the frequency domain of one image with the conjugate of the frequency domain of the other image and inverse transforming of the resultant image to the spatial domain [3]. It can also be shown that the convolution of two images in the spatial domain is exactly equivalent to the direct multiplication of the images in the frequency domain and inverse transform of the resultant image to the spatial domain.

The only difference between correlation and convolution is that in the frequency domain the complex conjugate[2] of one of the images is used in the multiplication for correlation where the direct pixel values are used for convolution operation. Since it is easier to explain the convolution process, it will be briefly discussed to provide an insight into why and when it is better to use spatial domain processing and when the equivalent frequency domain processing.

5.4.2- Convolution: Spatial or Frequency Domain?

Convolution in the spatial domain is often used for filtering images such as smoothing and derivative operations. To perform this filtering operation, a kernel of numbers is placed on the image and multiplied by overlapping pixels. Then the sum of the multiplications is placed at the pixel location covered by the central pixel in the kernel. The kernel is then moved to the adjacent location and the process is repeated until all of the possible kernel positions have been tried.

5.4.2.1- Kernel Size

Kernel sizes of 3×3 are quite common. Using such a kernel on a 512×512 image involves:

512×512= 262,144 Pixels or kernel locations

At each kernel location there will be:

3×3 = 9 multiplications and additions

which results in a total of

262,144×9 = 2,359,296 multiplications and additions

Sometimes larger kernel sizes are required which will result in much more processing. For example a 100×100 kernel used for the same image will result in:

262,144 × 100 × 100 = 2.6 Giga multiplications and additions

---

2 Conjugate of a complex number A+iB is A-iB which has the same magnitude but the phase has a different sign and has the effect of flipping the image about its vertical axis.

Even on very fast computers and careful coding of the operations, convolution using a 100x100 kernel will take a considerable amount of time. Special hardware may be used to speed up the process in the spatial domain.
Convolutions involving small kernels can be performed efficiently in the spatial domain. Performing the spatial domain convolution in the frequency domain involves the overhead of transforming both the image and the kernel into the frequency domain and then inverse transforming the result of the multiplication. As the size of the kernel becomes bigger, a point is reached where performing the operation in the frequency domain is more efficient despite of the overhead involved in the frequency domain operation.

5.4.2.2- Practical Differences

There are some practical differences between convolution in the spatial domain and multiplication in the frequency domain. In the spatial domain the size of the kernel is often much smaller than the image. In applying this kernel, the pixels that are closer than half the width of the kernel to the edge of the image are avoided since if the kernel is located at those regions some of the pixels in the kernel will not cover any pixels in the image. As a practical alternative, a different kernel that is one sided and has different weights can be applied near the edges. Another technique to overcome this problem is known as edge extension which stretches the edges out by the replicas of the edge pixels [9].

This edge problem will have a different manifestation in the frequency domain. A digital image is both finite in size and discrete due to sampling process and it is assumed to be periodical in both directions by the transformation process. For an image to be periodical it means that the edge pixels must have the same values so that if the image is wrapped around from top to bottom or from left to right the touching edges will have the same values and mark a period. The transformation process treats the image as a complete period in an infinitely long and wide virtual image. Therefore to minimise artifacts in the result the edge pixels must be set to equal values in both the image and the kernel. Note that the edge pixels of the image may be different from the edge pixels of the kernel.

In the frequency domain, the kernel must be the same size as the image since a direct multiplication of the corresponding pixels is performed. Also if the FFT or FHT is used for transformation[3] the size in pixels of both the image and the kernel must be a factor

[3] This is almost always the case since it is the fast implementation algorithms of the FFT or FHT which makes them attractive as a substitute for convolution in the space domain.

of 2. To achieve these requirements both the image and the kernel are padded with zeros or their mean pixel values to the nearest common power of 2 size.

5.4.2.2- Processing time comparisons

Table 5.4.1 shows the result of an experiment comparing the processing time spent to perform convolution in spatial domain and its equivalent in the frequency domain with different convolution kernels and image sizes. This experiment was run on a Macintosh PowerPC 7100 running at 66MHz with 28 Megabyte of memory.

	Image							
Size(pixels)	64x64		128x128		256x256		512x512	
	Processing time (ms)							
Kernel	Space	Frequency	Space	Frequency	Space	Frequency	Space	Frequency
3x3	150	1550	233	2500	550	6400	1700	21350
7x7	233	1550	520	2500	1520	6400	6350	21350
16x16	480	1550	1600	2500	**6700**	6400	**25450**	21350
32x32	1000	1550	**5070**	2500	22150	6400	92100	21350
50x50	**1550**	1550	10200	2500	49000	6400	204750	21350

Table 5.4.1- Processing time of spatial domain convolution and equivalent frequency domain operation with different kernel and image sizes.

Recall that for frequency domain convolutions both the image and the kernel must be the same size. Therefore in the above table, for the frequency domain processing kernels are enlarged to the size of the image and then convolved. Boldface numbers in the space column show the kernel size limits beyond which convolution will be faster if performed in the frequency domain. Comparison of the processing times reveals that for kernel sizes smaller than 50x50 convolution in the spatial domain is faster than frequency domain convolution.

5.4.3- Flow Diagram

As it was mentioned in section 5.2.1 some modules of the matching algorithm are common to all three methods of matching. In the following, only the operations which are specific to the frequency domain processing are discussed. A top level block diagram of the frequency domain matching algorithm is shown in Figure 5.4.2.

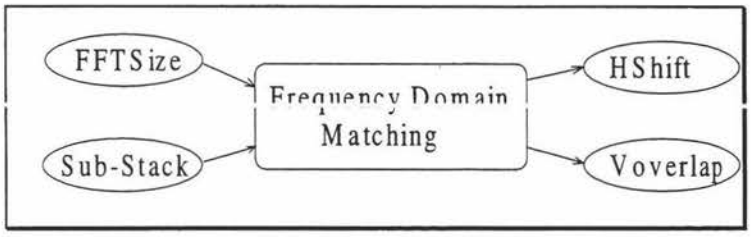


Figure 5.4.2- Top level diagram of Frequency Domain Matching

There are two inputs into the frequency domain matching: sub-stack and the FHTSize. The sub-stack consist of two images from the stack and the algorithm will find the match between them. The FHTSize is a power of 2 number and indicates the size of the selection from each image used in the frequency transformation. The output of the process is HShift which is the number of pixels that one image has to move in order to register with the other image in the horizontal direction and Voverlap which is the number of lines which one image has in common with the other image in the vertical direction.

Figure 5.4.3(a). shows the flow diagram of the processes performed on each image to obtain their frequency domain representation . Figure 5.4.3(b) shows the flow diagram of the processes performed on the transformed images produced by the previous process. This is to obtain cross correlation of them and also produce the outputs of the frequency matching process (HShift and Voverlap).

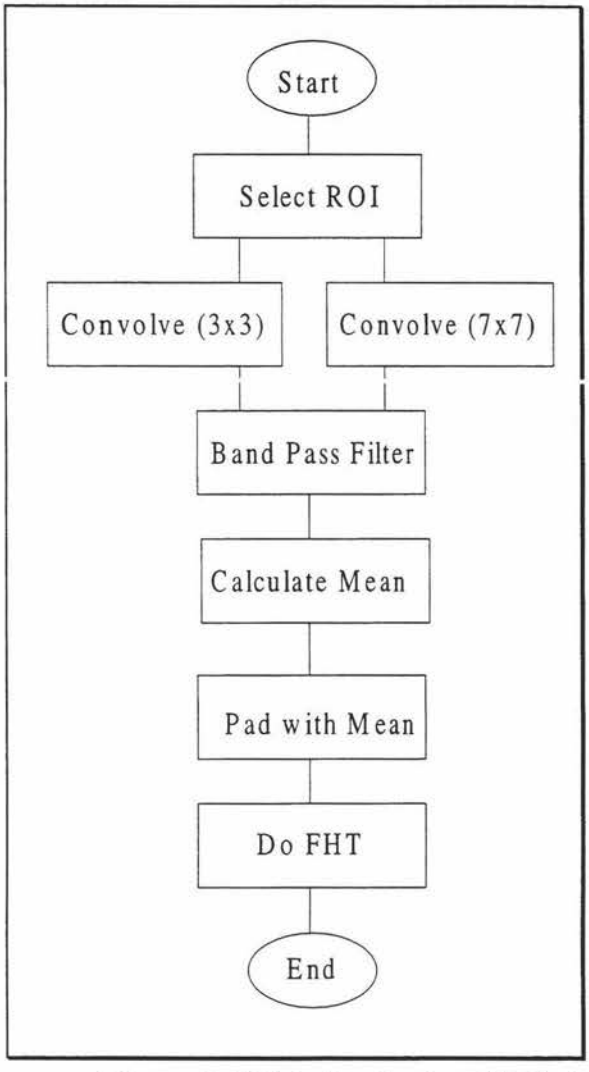


Figure 5.4.3(a)- Producing FHT of each image

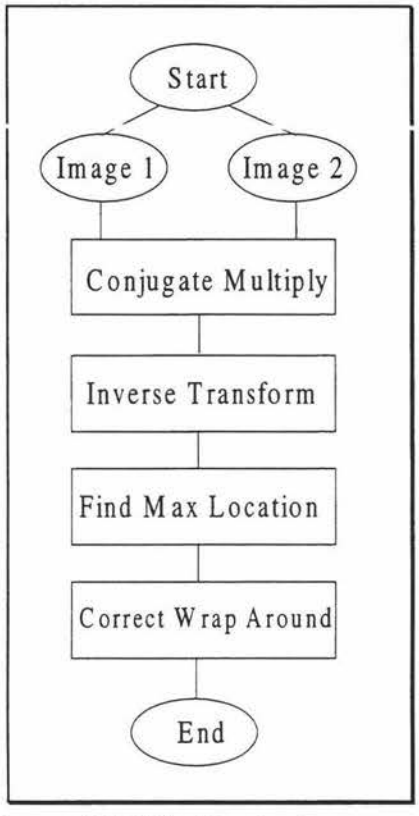


Figure 5.4.3(b)- Producing output

In the following paragraphs each step of the process is explained.

Select ROI- Only a part of each image is selected for finding a match in the other image. This is called a Region Of Interest (ROI). The size of ROI is determined by the FHTSize input to the algorithm. Figure 5.4.4 shows the selection of the ROI in an image. The ROI in the horizontal direction is taken from the middle of each image. In the vertical direction, ROI is taken from the top of one image and the bottom of the next image. These ROI's contain the overlaps between the images.

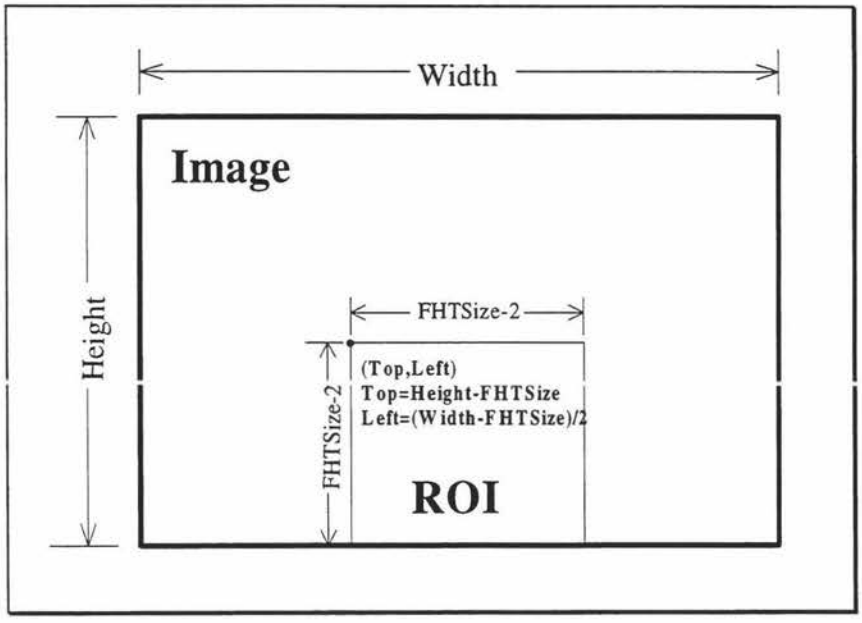


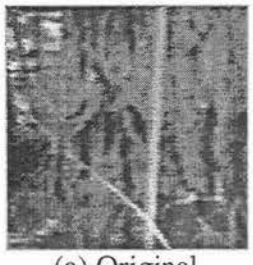
Figure 5.4.4- Selection of ROI in an image

Convolutions- There are two separate convolutions performed on the ROI. The size of one of the kernels used is 3x3 (see Figure 5.4.5) and the other is a 7x7 kernel whose values are all 1.

1	1	1
1	4	1
1	1	1

Figure 5.4.5- 3x3 Convolution Kernels

Both convolutions are done in the spatial domain and perform averaging of the pixels. The 3x3 kernel does averaging over 9 pixels (centre pixel and surrounding 8 neighbours) and the 7x7 over 49 pixels (centre pixel and surrounding 48 neighbours). The operation in effect is a Low Pass Filter. Sharper edges in the image produce higher frequencies and smooth edges produce lower frequencies. The smaller kernel allows higher frequencies to pass through than the larger kernel. Therefore the resultant image of the convolution with the larger kernel has smoother edges than the resultant image of the convolution with the smaller kernel. Both resultant images have smoother edges

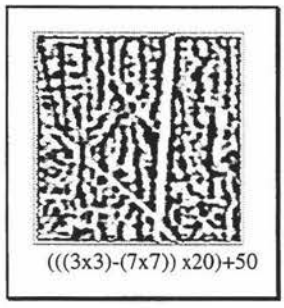


(a) Original

(b) 3x3 Filtered

(c) 7x7 Filtered

than the original image.


(((3x3)-(7x7)) x20)+50

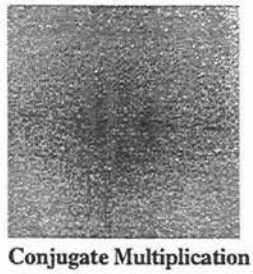
Band Pass Filter- The image resulted from the convolution with the larger kernel is first subtracted from the other image. This process of two convolutions with two different sized kernels and subtracting one from the other is equivalent to a Band Pass Filter operation on the original image [4]. The resultant image is an edge enhanced image and only contains those edges of the original image whose frequencies falls within the band pass of the filter. The lower and upper frequencies in the band are determined by the larger and the smaller kernels respectively. Since the pixel values in the resulting image are very small numbers, they are each multiplied by 20 and added to by 50 to increase the dynamic range of the pixel intensities. These constant values have been determined experimentally to produce satisfactory results for a range of images.

Calculate Mean- The mean value of the pixels in the resulting image is calculated to be used for padding the image to the FHTSize.

Pad with Mean Value- Since the FHT assumes the image is periodic, the edge pixel values must be the same at the top and the bottom and at the left and the right of the image. Note that the size of the selected ROI is 2 pixels less than the FHTSize in each direction. This allows one pixel of the same value to be added to each side of the image to bring its size to the FHTSize at both directions. The value of the pixels added to the edges is the average value of the whole image.

Do FHT- The resulting image is now transformed to frequency domain using Fast Hartley Transform algorithm.

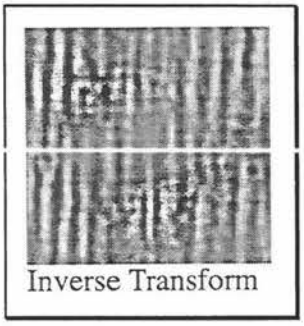
All the above series of operations are performed on both images in the stack. The resulting two images are then passed to the next process shown in Figure 5.4.3(b) and outlined in the following paragraphs.


Conjugate Multiplication

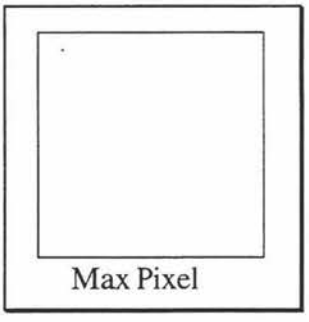
Conjugate Multiply- The conjugate values of each pixel in one images is multiplied by the corresponding pixel in the other image. Note that although only the magnitude of pixels is used in displaying the images in the frequency domain, all the operations on the image

uses the phase information as well as the magnitude information. The advantage of the frequency domain equivalent of the correlation in the spatial domain is this simple and few pixel by pixel multiplication as opposed to millions of multiplication and additions required in the spatial domain [10].

Inverse Transform- To get the equivalent to the correlation in the spatial domain the resultant image has to be retransformed to the spatial domain.


Inverse Transform

Find Maximum location- The location of the maximum value pixel in the correlation image marks the best match between the two images in the stack. Coordinates of the maximum value pixel are indications of the Hshift and Voverlap.


Max Pixel

Correct Fold-Over - There is a foldover effect in the correlation image as a result of the technic used to speed up the process. This effect is explained in section 5.4.4.

Priori knowledge of the characteristic of the system is needed to correct for the fold over effect. In the horizontal direction, Hshift can have both positive and negative values typically smaller than 10. This indicates that one image has to be moved to the left or to the right respectively for a match. However Voverlap is always positive typically around the FHTSize value and indicates that one image has to always move downwards for a match. Once the coordinates of the maximum pixel in the correlation image are found, if the horizontal coordinate is larger than 1/2FHTSize, using the priori knowledge of the typical Hshift values, it is concluded that the coordinate must have been a negative value folded back over the image from the right. Therefore it is corrected for by subtracting FHTSize from it. If the vertical coordinate of the match point is smaller than 1/2FHTSize, then it is concluded that it has been folded over from the bottom. Therefore the coordinate is corrected for by adding FHTSize to it.

5.4.4- Processing Time

Recall that to perform correlation in the spatial domain one image is placed on top of the other image and overlapping pixels are multiplied etc. The image is then moved to a new location and the process is repeated until all the possible locations are visited. If images A and B of the size nxn are to be correlated, image B has to be padded with

zeros or any other suitable value to four times its dimension (2nx2n) so that for every placement of the image A on top of the image B there will be some pixels to multiply with (see Figure 5.4.6). Therefore the image resulting from this correlation will be of the size 2nx2n. An alternative explanation for padding the images is that since every pixel in the correlation image corresponds to a shift vector, and there are 2nx2n possible shift vectors for two images of the size nxn, the correlation image must be of the size 2nx2n to hold the information. Since frequency domain processing is an exact equivalent to the spatial domain correlation the equivalent frequency domain processing should also produce a result of size 2nx2n. For this to happen each image has to be padded so that its size is 2nx2n (assuming that n is a power of 2) before transforming to the frequency domain.

The processing time for frequency transformation is directly proportional to the size of the image. Increasing the size of the image to four times its original size will increase the processing time by a factor of four. Since in the frequency domain processing both images need to be enlarged, the processing time will increase by a factor of eight. To avoid this processing time increase, in this application images are left at their original sizes. Recall that in the FFT or FHT method it is assumed that an image is a single period of an infinitely large and periodic image. Leaving an image at its original size before transformation gives rise to a foldover effect in the resulting correlation image. This is illustrated in Figure 5.4.7.

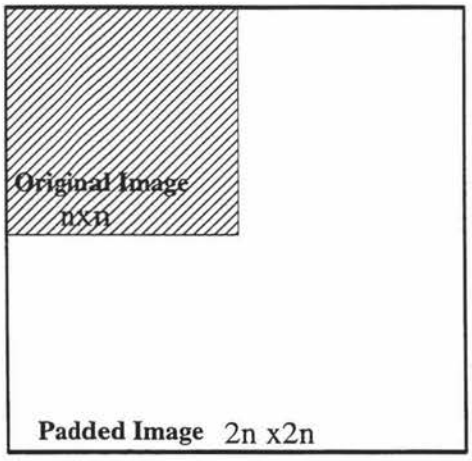


Figure 5.4.6- Image padded to 4 times its original size

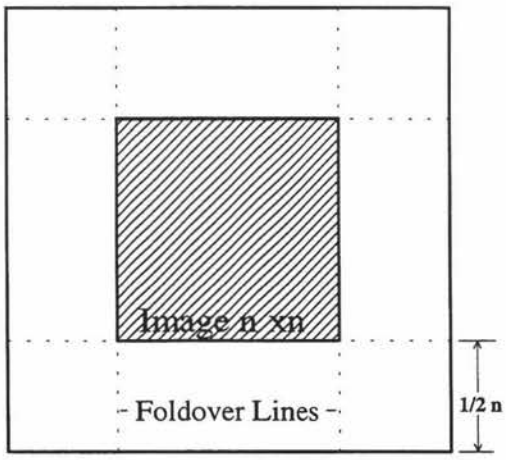


Figure 5.4.7- Fold over effect

It appears that the surrounding n/2 pixels are folded back into the image. The implication of this is that the maximum point in the correlation image (which indicates the position of the match between the two images) may have been in the surrounding

area and is folded back into the image. This offset has to be corrected for in the foldover correction part of the algorithm. Since the only variable effecting the time to find a match using the frequency domain algorithm is the FHTSize, this technic has speeded up the operation of the correlation in the frequency domain by a factor of 8 times.

Table 5.4.2 tabulates the total processing time to find a match between two images using different FHTSize for operation.

FFTSize(Pixels)	Time(ms)
32 x 32	2200
64 x 64	2480
128 x128	3450
256 x 256	7500
512 x 512	23000

Table 5.4.2- Processing times for different FHTSizes

Note that there is a constant overhead time in all of the processes of approximately 2000ms. This is a result of the operating system activities and initial processing that is carried out by the program for all frequency transform operation regardless of the size of the ROI.

5.4.5- Time versus Accuracy

Selection of a smaller ROI would save mosaicing time but also may reduce the accuracy of the result. With a smaller selection there are fewer pixels available for matching. The size of the ROI used in transformation and matching must be suitable for the amount of horizontal shift and the vertical overlap that exists between the two images. Experiment shows that kernel sizes smaller than 32×32 reduce accuracy to unacceptable levels. Most accurate results will be obtained when the size of the ROI is close to the vertical overlap. This is assuming that the horizontal shift is significantly less than the vertical overlap (about 10%) which is almost always the case. This accuracy comes about because when the size of the ROI selection is close to the vertical overlap, then the two ROI selections from the images include most of the overlaps between images and can be well matched. Only a small portion of the ROI selections will be different. This will give a very sharp peak at the match point in the correlation image. This peak is what is searched for in the correlation image to determine the location of the best match.

5.4.6- How Successful this algorithm is?

The most critical factor affecting the accuracy of the results is the size of the ROI selection needed for matching.

If the FHTSize is too large there will be two potential problems. One is that given a specific FHTSize, the range of the vertical overlap that can be found is 1/2FHTSize to 3/2 FHTSize. If the overlap in outside this range it can not be found correctly. The other problem is that when the FHTSize is much larger than the vertical overlap (but still within the correct range) only a small part of a ROI selection is the same as the ROI selection of the other image and most areas are different. This has the result that there will not be a sharp peak in the correlation image and it is occasionally possible to get an incorrect peak and therefore an incorrect indication of a match.

If the FHTSize is too small there will also be two potential problems. One is that the small ROI selections from the two images may not include any common areas of the images and therefore there can not be any correct match found between them. The other potential problem is that since there are only few pixels involved in the operation and there is a also the likelihood that there is only a small area of common features in the ROI selections, an incorrect match might be found.

However if a suitable FHTSize is chosen for the images, then the results are extremely good, no failure was observed in 20 sample images. To test the robustness of the process, some randomly chosen images were modified. The modification was performed on one of the images in a pair to be matched and the other image was left in its original form. The modifications included changing the brightness, contrast, sharpness, scale and rotation. There were no failures in the case of change in brightness, contrast and sharpness. The process is, however, sensitive to scaling and rotation of the images. Fortunately there is no relative scaling or rotation between consecutive images produced by the scanning mechanism and therefore this weakness is of no concern for this application.

6- COMPARISON OF THE DIFFERENT ALGORITHMS

6.1- Strip Matching

Advantages:

- If the size of strip is small it is the fastest algorithm;
- The success of this algorithm is not dependant on the actual number of overlapping lines in the two images. This is because the strip is generally much smaller than the number of overlapping lines and is almost always within the overlapping areas in the two images. Problem occurs if the strip is larger than the actual overlapping lines and therefore can not fully exist within the overlap.

Disadvantages:

- For better results size of the strip must be large which slows down the process;
- If the bottom of an image is plain and without much significant features, the algorithm is likely to produce incorrect results;
- The values of Hrange and Vrange are critical to the success of the algorithm but are not always known. These values limit the search range in order to save processing time however they must be large enough to ensure that the matching strip in the other image is within the search range and will be analysed.

6.2- Block Matching

Advantages

- Unlike strip matching algorithm that blindly chooses the strip to find a match for, it uses a block with significant features for a match and therefore is a better algorithm for plain images.

- The success of this algorithm is not generally dependant on the number of overlapping lines in the images since the block size is typically smaller than the overlapping lines. This algorithm is generally more robust than the strip matching algorithm except when the number of overlapping lines is smaller then the block size.

Disadvantages

- It is generally slower than the strip matching algorithm since it involves additional process of finding a block with significant features.

- In addition to the problem associated with the values of Hrange and Vrange common with the strip matching algorithm, it suffers from dependency on the Range value as well. The value of Range must be small enough so that the block found in one image falls in the overlapping area of both images. Since the number of overlapping lines is not known in advance, Range value must be chosen small which in turn increases the probability of incorrect result.

6.3- Frequency Domain Matching

Advantages

- It is faster than other algorithms where big selection of one image is searched for a match in the second image;
- Since a larger selection of one image is searched for a match in the other image it generally produces the most accurate results;

Disadvantages

- The number of overlapping lines must be between 0.5 and 1.5 of the size of the selection from the images used for frequency transformation. If a wide range of number of overlapping lines is possible in different images, then this method can not be used reliably.

7- ROBUST ALGORITHM

The mosaicing algorithm must satisfy two requirements. One is that it should have a high success rate in excess of 90% . The other which is more important is that if a result is likely to be incorrect it should be recognised by the algorithm and the user be warned. In other words the mosaicing process has to be fail safe.

7.1- High Success Rate

To satisfy the first requirement a new algorithm was devised that combines the strengths of the previous algorithms. The frequency domain processing proved to produce the best results if the FFTSize is chosen appropriate to the actual size of the overlap. Therefore if a suitable FFTSize can be found with a different algorithm which is not sensitive to the actual size of the overlap and used for frequency domain processing, best results can be expected.

The strip matching process is the algorithm least sensitive to the actual size of the overlap, however it does not produce very reliable results if the bottom of the image to be matched does not contain significant features. The block matching algorithm is more robust for plain images but is more sensitive to the actual size of the overlap than the strip matching algorithm.

The best combination of algorithms for "busy" images is to find the overlap between the first and the second images in the stack using the strip matching algorithm and then use this overlap to choose a suitable FFTSize for the frequency domain processing to find all the overlaps between all the images in the stack including the first and the second images. Since tree images are busy images with lots of edges corresponding to branches and leaves, the strip matching algorithm is used as preprocessing algorithm.

The best combination of algorithms for plain images is the initial use of the block matching algorithm followed by the frequency matching algorithm as above.

7.2- Choosing Suitable FFTSize

Recall that for the frequency domain processing if the FFTSize=n then the overlap that can be detected is between 0.5×n and 1.5× n. Table 7.1 summarises the relationship between different FFTSizes and the related overlap range that can be detected.

FFTSize	Overlap Range
32	16---48
64	32---96
128	64---192
256	128---384

Table 7.1- FFTSizes and related overlaps

It can be seen in the Table 7.1 that some overlaps can be detected by two different FFTSizes. For example an overlap of 35 lines can be detected by both FFTsizes of 32 and 64. To choose a suitable FFTSize for the overlap the midpoint of the maximum limit of one FFTSize and the minimum limit of the next FFTSize is chosen as the boundary as shown in Figure 7.1.

Figure 7.1- Overlap limits for suitable FFTSizes

$$\left.\begin{matrix}32\\64\end{matrix}\right\} \quad \frac{48+32}{2}=40$$

$$\left.\begin{matrix}64\\128\end{matrix}\right\} \quad \frac{96+64}{2}=80$$

$$\left.\begin{matrix}128\\256\end{matrix}\right\} \quad \frac{198+128}{2}=160$$

For example if the overlap found by strip matching or block matching algorithm is less than 40 then an FFTSize of 32 will be used for the frequency domain processing otherwise an FFTSize of 64 will be used.

Using this algorithm on 20 sample tree images has produced 100% correct results.

7.3- Fail Safe

The algorithm should warn the user if there is a likelihood that one or more of the matches are incorrect. There are two different and complementary checks that is performed on the results to validate them. One is by considering the characteristics of the system as a whole and the other considering the characteristic of images within a stack.

Systems Characteristics- In Section 5.2.3 it was discussed that examination of 20 images showed a maximum Voverlap variation of ±18 lines. This figure was obtained from images taken by the system at different distances between the camera and the base of the tree up to 20 meters. The Hshift value for the above is ±3 if the camera is secured properly on the tripod. Considering these figures, in the worst case situation the Voverlap value varies between each consecutive image by 18 lines. Therefore the maximum allowable variance for the Voverlap is:

$$\begin{aligned}\delta^2 &= \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2 \\ &= \frac{1}{n}\sum_{i=1}^{n}(18/2)^2 \\ &= 81\end{aligned}$$

Likewise the maximum allowable variance for the HShift is $(3/2)^2 = 2.25$.

Once all the Hshifts and Voverlaps between images in a stack are determined, then the mean and variance of the found results are calculated for both the Hshift and Voverlap. If either of the calculated Hshift and Voverlap variances are greater than the maximum allowable variance then the results may not be correct and a warning is issued.

Input Images Characteristics- It is possible that most of the match results between images in a stack are very close together except one or two. These type results may be incorrect but will produce a variance which may be less than the maximum variance and therefore not detected by the above method. To single out a likely incorrect match between two images in a stack once the stack variance is found, the square of the difference between one match result and the next is compared against the stack variance and if it is greater than 4 times (twice the Standard Deviation corresponding to 97.8% under the normal distribution bell) then a warning is issued.

8- SUMMARY

An application of digital imaging technology is developed to obtain objective measurements of tree parameters for preharvest inventory assessment for the forestry industry. To obtain high resolution images suitable for this task a number of partially overlapping low resolution images are scanned from successive sections of a tree. The successive overlapping images need to be registered, their overlaps removed and the remaining stitched together to construct the high resolution image. The objective of this work was to develop algorithms to achieve this.

Algorithms

Algorithms developed here fall into two broad categories: Spatial Domain Processing and Frequency Domain Processing.

- **Spatial Domain Processing**- Two technics were developed in the Spatial Domain: Strip Matching and Block Matching.

  a)- Strip Matching- To find the match between images 1 and 2 a strip of pixels is taken from the bottom of image 1 and is compared with the strip of the same size from the top of the image 2. The absolute sum of differences between individual pixels in the strip is then calculated. This strip is then moved around to the left and right to find the horizontal shift and down to find the vertical overlap. The same operation is repeated for each strip location. The location with the minimum absolute sum of differences is the best match (Section 5.2).

  b)- Block Matching- With the strip matching method it is possible that the strip does not contain any significant feature to be matched to in the other image. Block matching solves this problem by first searching on image 1 to find a small block of pixels which contain the most significant features. Then this block is used for matching (Section 5.3).

- **Frequency Domain Processing-** With both the strip and block matching methods a very small fraction of the actual overlap is taken from image 1 for finding a match in image 2. In Frequency Domain Processing a bigger block of pixels is used for matching. Cross correlation is performed in the frequency domain between selected blocks from the overlapping images. The location of the maximum pixel in the cross correlation image indicates the best match (Section 5.4).

Results

1) - Strip matching is the fastest algorithm if the strip size and search ranges are small (see section 5.2.3). It is not, however, a suitable algorithm for plain images without many edges and significant features.
2) - Block matching algorithm is slower than strip matching algorithm but is more suitable for plain images (see section 5.3.2). The drawback of this algorithm is that the search range value to find the block must be smaller than the overlaps between images; an unknown value at the beginning of the algorithm.
3) - Cross correlation performed in the frequency domain produces the best results in a reasonable time for all images (see section 5.4.4). The weakness of this algorithm is that it requires the overlaps to be between 0.5 and 1.5 of the size of the ROI selected from images for matching, again an unknown value at the beginning of the algorithm.

Final Algorithm

100% correct results were obtained with cross correlation algorithm performed in the frequency domain provided that the correct size of ROI was selected for the processing. To find the correct size of ROI, strip matching algorithm which is not sensitive to the overlap size is used first. Then this ROI size is used for frequency domain processing (see Section 7.2).

Fail Safe

The result of the matching is fail safe. The final algorithm warns the user if there is a likelihood that one or more of the matches are incorrect. This is achieved by examining the variance of the results (see Section 7.3).

9- REFERENCES

[1] **Rasband, W.** 1993, NIH-Image User Manual, Published On NIH-Image Home page on the Internet.

[2] **MaxVideo 20 Hardware Reference Manual (1993),** *DataCube Inc,* 300 RoseWood Drive, Danvers USA, Document No. HW599-2.1

[3] **Jack, Keith,**1955, *Video demystified: A hand book for the digital engineer,* HighText Publications, Inc. ISBN 1-878707-09-4

[4] **Anil k. Jain,** 1989, *Fundamentals of Digital Image Processing,* Prentice Hall Inc.

[5] **Cooley, J. W. and Tukey, J.W.** ,1965, *An Algorithm for the Machine Calculation of Complex Fourier Series,* Mathematics of computation ,1965

[6] **Hartley, R. V. L.** ,1942, *A more symmetrical Fourier Analysis Applied to Transmission Problems,* Proc. IRE, March 1942

[7] **BraceWell, R. N.,** *The Fast Hartley Transform,* Proc. IEEE. Vol. 72, No. 8, August 1984.

[8] **Reeves A. A.** 1990, *Optimised Fast Hartley Transform for the MC68000 With Applications in Image Processing,* Thayer School of Engineering, Darmouth Colledge, New Hampshire

[9] **Russ, J. C.** 1992, *The Image Processing Handbook*, CRC Press

[10] **Strang, Gilbert,** 1986, *Introduction to Applied Mathematics,* Wellesley Cambridge Press

[11] **Vivino, M.** 1993, *Inside NH-Image Manual,* Published On NIH-Image Home page on the Internet.

[12] **Swidbert, R.** 1995, *AutoMatch,* Published On NIH-Image Home page on the Internet.

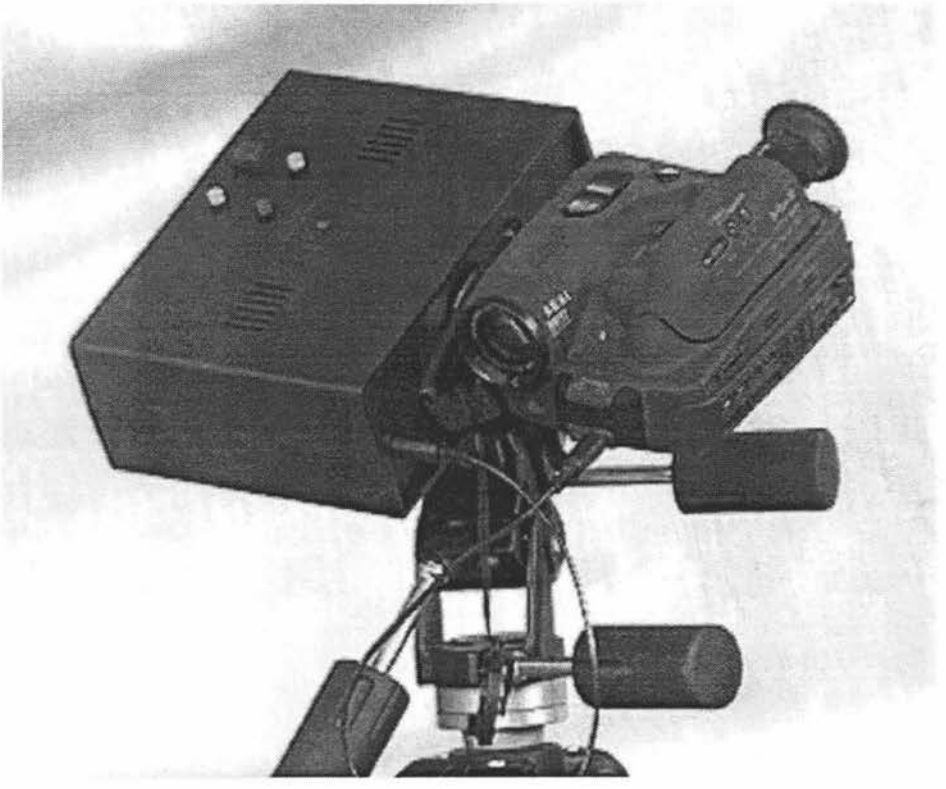
[13] **Weehuizen, M.** 1996, *Development of an in-field Tree Imaging System,* Department of Production Technology, Massey University.

[14] **Overington, I.** 1992 , *Computer Vision- A Unified Biologically-Inspired Approach,* Elsevier

APPENDIX A- VIDEO TAPE BASED HIGH RESOLUTION IMAGING SYSTEM

High Resolution Videotape Based Computer Imaging System

Image Acquisition Phase
Design and Development Report



Farshad Nourozi
March 1995

Department of Production Technology
Massey University
Palmerston North
New Zealand

Table Of Contents

1- INTRODUCTION

Tasman Forestry routinely makes pre-harvest assessment of mature radiata pine trees. Data collection for the assessment is performed by looking at a standing tree in a sample plot and estimating measurements such as height of the tree, diameter of the stem at different heights, sweep of the stem, features along the length of the tree, etc.

In 1993, The Department of Production Technology carried out a feasibility study for Tasman Forestry on the use of image processing for automation of pre harvest inventory to provide more objective measurements. Part of the study was concerned with determining a method of capturing high resolution images (better than 500×8000) of trees and their transfer to a computer. Once captured some tree parameters and measurements could be determined to within ±1cm accuracy.

One proposal was to use a videotape camera and a scanning mechanism to capture a series of images which could be latter transferred to a computer and combined to form a single high resolution image.

This proposal is implemented in two phases, image acquisition phase and image analysis & construction phase. Phase 1 (image acquisition) is mostly hardware based and is concerned with the design and construction of the image capturing unit. Phase 2 (image analysis & constructions) is all software based and is concerned with analysing a series of the acquired low resolution images and constructing a single high resolution image from them.

This document describes the implementation of the phase 1 of the proposal.

2- IMAGE CAPTURE

The image capture system consists of :

- two dimensional array based CCD video tape camera;
- scanning device mounted on a tripod;
- decoder;
- frame grabber card installed in a personal computer;
- software to manage the acquisition of frames into the computer and;
- software to process the frames to form a full image.

The operation and characteristics of each of these components are described in the following paragraphs:

2.1- Camera

An "off the shelf", high quality video camera recorder (AKAI PV-MS8) was chosen to record the images on to 8mm video tape. Some important specifications of the camera are as followings:

- The camera uses PAL color, Hi-8 video system technology. The Hi-8 video system is based on the 8mm video system with some improvements. The main advantage of the Hi-8 system is its increased bandwidth over that of ordinary 8mm technology. This is achieved by shifting the FM carrier frequency for the luminance signal from the 4.2-5.4MHz range to the 5.7-7.7MHz range. This increases the horizontal resolution to more than 400 lines.

- S-Video output- The S-Video output transmits the video signal as two separated components, the luminance component and the chrominance component. The luminance component is the black and white information about the image and the chrominance component is the color information about the image. Separation of the video signal minimises flicker and color blur in the picture and enhances the sharpness of the picture. The chrominance component of the signal is not required for this system and is therefore discarded.

- The camera uses a 1/3″ CCD image sensor with 420,000 sensor elements arranged in a rectangular grid of rows and columns and an 8× power zoom lens (F/1.8, f=6.0 - 48mm).

- The minimum illumination required for camera operation is 3 lux, however the recommended illumination is 100 lux.

- Aperture and shutter speed can be controlled manually or automatically. Aperture settings from F1.8 to F16 in 7 steps and shutter speeds from 1/50s to 1/10000s in 8 steps are available.

- Focus can be done manually or automatically.

2.1.1- Resolution

The system is required to produce images which allows diameter measurements accurate to ±1cm and height measurements accurate to ±50cm.

The sensor elements on the camera are arranged in a rectangular grid of rows and columns. The information from these sensors is transferred to the tape a line at a time as an analog signal. The format of this analog signal is bandwidth limited in such a way that about 400 distinct values per line are retained. One complete frame consists of 625 lines of which only 575 are active (574 complete lines plus a half line at the top and another half line at the bottom). The rest of the lines are used for vertical retrace timing.

Images recorded in the above format are later played back and a frame grabber is used to convert this analog video signal to digital signals. In this process only 512 lines out of 575 active video lines are captured and the rest is discarded. In the horizontal direction the signal is over sampled to produce 768 discrete pixel values per line. This means that some adjacent pixels will have been derived from essentially the same information. For this reason in calculating resolution in the horizontal direction it is assumed that only 400 pixels are available per line, even though 768 pixels form a line in the image.

With the camera located at 15m from the tree and with full zoom for a close up view, the image includes about 1.33m×1m of the object in the field of view. Therefore the angle of the field of view in vertical direction for this set up is:

$$\mathrm{Tan}^{-1}\theta = 1/15$$

$$\theta = 3.81°$$

Having 1m of the object in the field of view in the vertical direction results in a vertical resolution of:

$$100cm \,/\, 512 Lines \quad \text{or} \quad 0.2cm \,/\, Line$$

at the base of the tree. This resolution allows measurements of features as small as 0.2cm with a possible error of ±0.1cm:

$$\pm 0.2(cm \,/\, Line) \,/\, 2(Lines) = \pm 0.1cm$$

since a minimum of 1/2 line is required for the smallest feature to be measured. (required accuracy in the vertical direction is only within ±50cm).

Angle of view in the horizontal direction for the above set up is:

$$\mathrm{Tan}^{-1}\alpha = 1.33/15$$

$$\alpha = 5°$$

Having 1.33m of the object in the field of view in the horizontal direction results in a horizontal resolution of:

$$133cm \,/\, 400 Lines \quad \text{or} \quad 0.33cm \,/\, Line$$

at the base of the tree. This resolution allows measurements of features as small as 0.33cm with a possible error of ±0.17cm:

$$\pm 0.33(cm \,/\, Line) \,/\, 2(Lines) = \pm 0.17cm$$

As the camera scans up the tree, this resolution decreases because the angle of the field of view remains constant at 3.81° in the vertical direction and at 5° in the horizontal direction while the distance between the camera and the stem of the tree within the field of view increases (Figure 1). For example the vertical resolution at the height of 30 meters up the tree is:

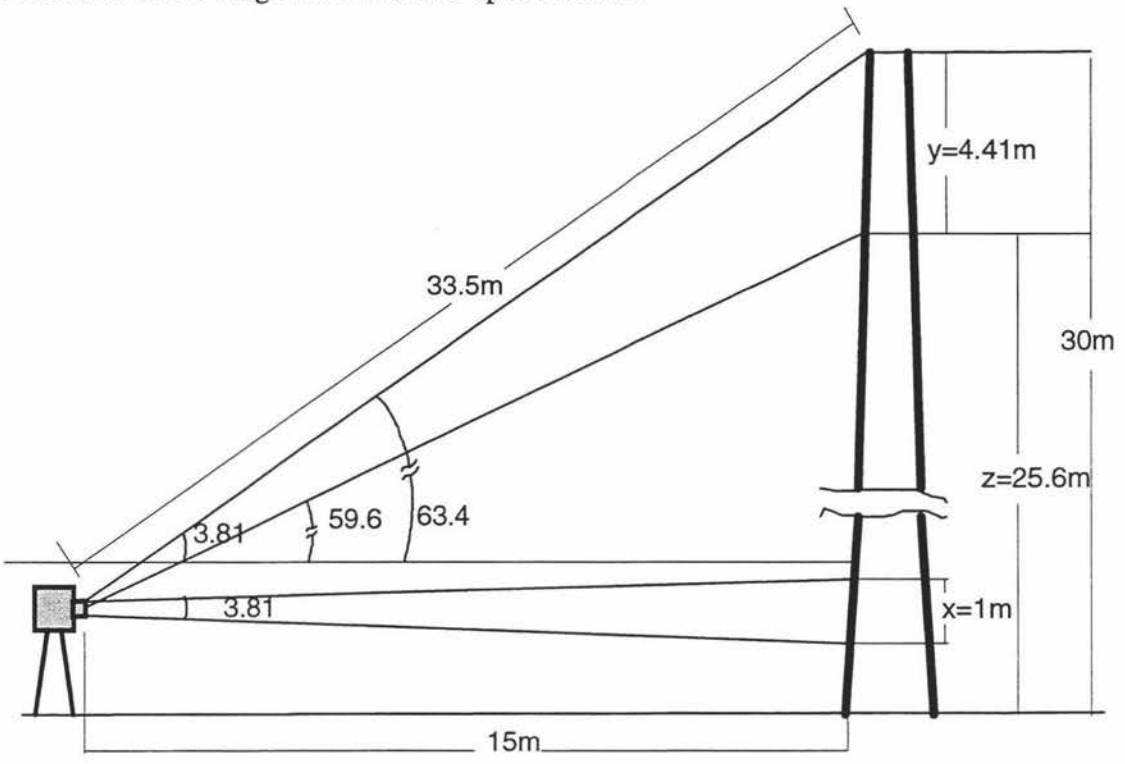


Figure 1- Tree within the field of view at different angles.

$$\theta' = \text{Tan}^{-1}(30/15)$$
$$= 63.4°$$

$$\theta'' = 63.4° - 3.81°$$
$$= 59.6°$$

$$Z = \text{Tan}\,\theta'' \times 15$$
$$= 25.6\text{m}$$

$$y = 30 - 25.6$$
$$= 4.41\text{m}$$

Therefore the maximum vertical resolution obtained at the height of 30m is:

$$\frac{4.41m}{512Lines} = 0.86(cm / Line)$$

This will enable the image processing system to measure features as small as 0.86cm in the vertical direction with a possible error of ±0.43cm:

$$\pm 0.86(cm / Line) / 2(Lines) = \pm 0.43cm$$

at the height of 30m up the tree.
Resolution of the image also decreases in the horizontal direction as scanning proceeds to the top of the tree. At the height of 30m up the tree distance between the camera and the segment of the tree in the field of view of the camera is:

$$\sqrt{15^2 + 30^2} = 33.54m$$

Therefore the field of view of the camera in the horizontal direction covers:

$$Tan5^\circ \times 33.54m = 2.93m$$

of the width of the tree. This will result in a horizontal resolution of about:

$$293cm / 400Lines \quad \text{or} \quad 0.73cm / Line$$

This resolution enables the image processing system to measure features as small as 0.73cm in the horizontal direction with a possible error of ±0.37cm:

$$\pm 0.73(cm / Line) / 2(Lines) = \pm 0.37cm$$

at the height of 30m up the tree.

Table 1 summarises resolutions at different heights when the camera is set up 15 meters from the base of the tree:

Height(m)	Vertical Res(cm/Line)	Horizontal Res(cm/Line)
1	0.2	0.33
30	0.86	0.73

Table 1- Resolution summary

Note that the resolution in the vertical direction degrades more than the resolution in the horizontal direction as the height increases.

Higher resolutions could be obtained by this system if either the camera was positioned closer to the tree or a more powerful zoom lens was used. However because the field of view in the horizontal direction would then not cover the whole width of the tree, a two dimensional scanning scheme would be required.

2.2- Scanning platform

Capturing an image which is larger than the field of view of the camera is achieved by using a scanning platform. The camera and the scanning platform mounted on a tripod are shown in Figure 2.

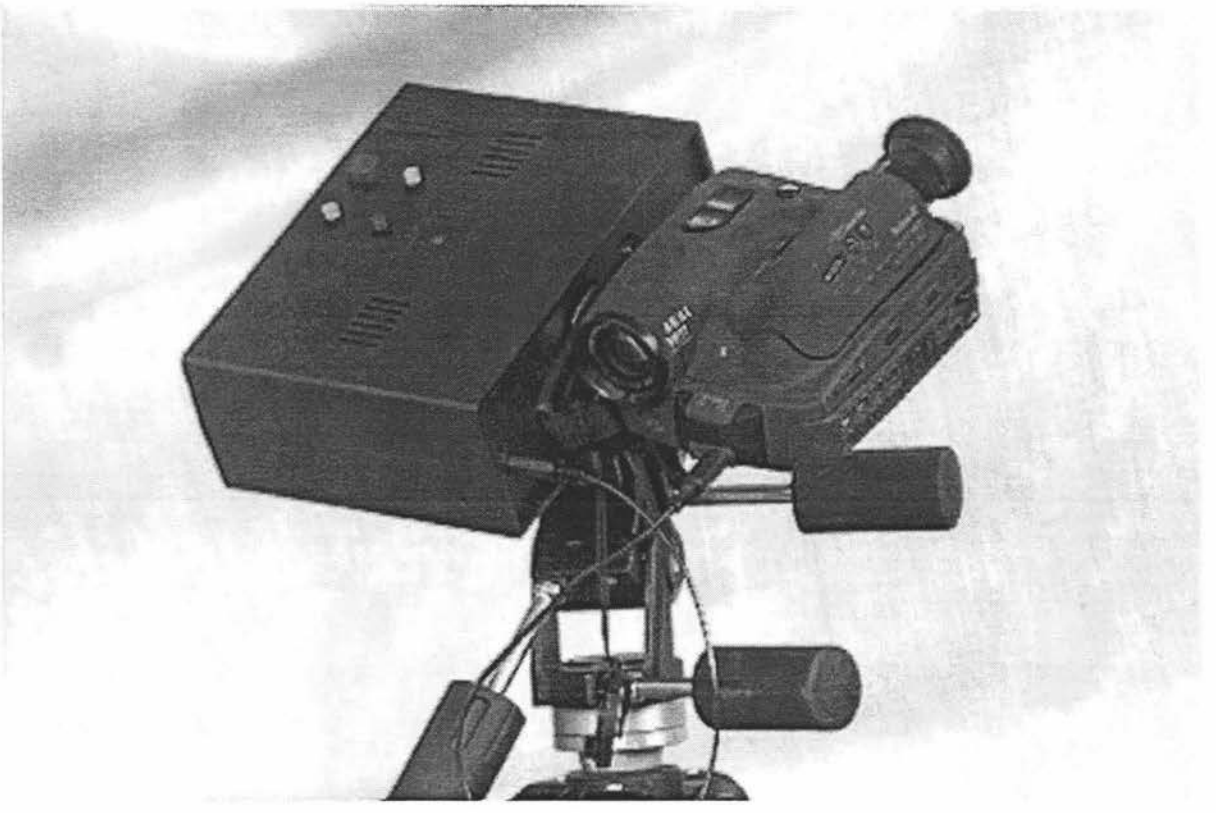


Figure2- Camera and scanning platform on tripod

The camera is mounted on the scanning platform which rotates through 75.6° in the vertical direction. This angle of rotation will ensure that if the camera is 10m or more from a 40m tree, it can be scanned to almost its full height:

$$10m \times Tan75.6° = 38.9m$$

In traversing through this angle the scanner stops at fixed angle intervals to allow the camera to capture several frames. This ensures that blurring of the image does not occur due to movement and also the camera is given sufficient time at each position to automatically adjust the brightness and focus of the images. The camera captures images at the rate of 25 frames per second during the whole duration of the scan. Only few of these frames are required to make up a complete image.

After about 0.5 second following each stop, an audio signal at 1KHz is generated and recorded on the audio track of the tape. This 0.5 second delay is required to allow stabilisation of the mechanical vibration of the camera due to a sudden stop. It also gives enough time for the camera to automatically adjust the focus and the exposure time for the new position.

The audio signal effectively tags a frame as valid. The valid frames will be digitised and transferred to the computer. The rest of the frames will be ignored. To be able to put the individual frames together and make a full image of the tree, each valid frame need to have some overlap with the adjacent valid frames. When the camera is fully zoomed for close up view, the angle of view in the vertical direction is 3.81°. A smaller angle between stops is required to ensure an overlap between adjacent frames. Therefore an angle of 3.6° between steps was chosen which provides about 5.5% overlap. A full scan is therefore completed in $75.6° / 3.6° = 21$ steps. Each step consists of moving from the previous position to the next (400ms) and a stabilisation wait period before the generation of the audio signal (500ms) and the time that the audio is being generated (250ms). This provides a scan time for a tree is less than 25 seconds. This time could be reduced to 10 to 15 seconds if mechanical vibration could be eliminated. Once the scan is completed it takes 8.4 seconds for the scanner to return to the home position.

Components of the scanning platform are described in the following sections.

2.2.1- Hardware

The scanning platform consist of:

- a stepper motor;
- Home position sensor;
- gearbox;
- camera mount pad;
- controller board;
- batteries;
- Input\Output Connectors:
  - battery charger input;
  - audio output;
  - power output;
- switches and indicators for user operation.

The whole unit is mounted on a tripod.

2.2.1.1- Stepper Motor

A 4 phase hybrid stepper motor was chosen to drive the camera from one position to the next (Figure 3).

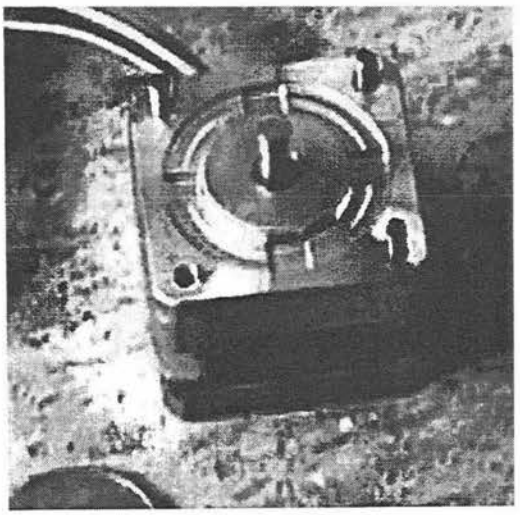


Figure 3- Stepper Motor

Using a stepper motor allows an accurate control over the camera position and speed of the movement. The step angle of the motor is 1.8° per step (ie. 200 steps/revolution) and the positional accuracy is 5-10% of one step angle (ie. 0.09-0.18°) which is non-accumulative and therefore remains constant regardless of the number of steps taken. Being a 4 phase motor, this error averages to zero in a 4 step cycle. Therefore to obtain the best accuracy, the motor is driven through multiples of 4 steps from one position to the next (ie. 100 steps) . The stepper motor can be stepped as fast as 880 steps per second, which results in a speed of:

$$\frac{880(steps\ /\ \sec ond)}{200(steps\ /\ rev)} = 4.4(rev\ /\ Second)$$

However at this speed, starting torque provided by the motor is very small (ie. nearly zero). The motor is stepped at 250 steps/second (ie. 1.25 Rev/S) which provides a torque of about 300mNm.

When the motor windings are not energised the detent torque* of the motor (30mNm) is enough to stop rotation of the motor as the result of the torque exerted by the load.

Operating voltage of the motor is 5V and it draws a maximum current of 500mA. The current may be reduced by the driving circuitry however this would degrade the torque/speed characteristic of the motor.

2.2.1.2- Gearbox

The shaft of the stepper motor is coupled by a gearbox to the camera mounting pad (see Figure 5). The gearbox is a worm gear arrangement and consists of two gears which provide a speed reduction of 50:1 . This gear ratio was calculated as following:

Time to move from one frame position to the next: $t_f = 400ms$
Step frequency: $f_s = 250$
Steps/Second
Stepper motor resolution: $R = 1.8°/step$
Camera movement from one frame to the next: $\phi = 3.6°$

Motor angular speed:

$$\varpi = f_s \times R$$
$$= 250 \text{ Steps/Second} \times 1.8°/\text{step}$$
$$= 450°/\text{Second}$$

Motor angular movement in 400ms:

$$\theta = t_f \times \varpi$$
$$= 400ms \times 450°/\text{Second}$$
$$= 180°$$

$\Rightarrow$ Required gearbox input to output ratio:

$$\text{Ratio} = \theta / \phi$$
$$= 180° / 3.6°$$
$$= 50/1$$

The gearbox also provides a 90° shift between the plane of rotation of the motor and the plane of rotation of the camera mounting pad.

2.2.1.3- Camera Mount Pad

The camera is mounted on a pad directly connected to the gearbox output shaft (see Figure 5). This locates the camera at the side of the unit's enclosure thus providing a clearance to move through 90°. The mount pad has a standard tripod bolt which screws into the tripod mount base of the camera.

* Maximum torque that can be applied to the spindle of the motor when the windings are not energised, without causing continuous rotation.

2.2.1.4- Home Position Sensor

The position of the camera can be monitored by the microcontroller all the time. This is achieved by using a sensor which detects a reference point (ie. home position). The operation of the sensor is as following:
The sensor which includes an infra red emitter and detector is pointing to a reflective disk from which a slice is cut off (See Figure 4). When the solid part of the disk is in front of the sensor, the infra red is reflected off the surface and is detected by the detector. The output of the detector in this case is about zero volt. When the part of the disk which is cut off is in front the sensor, there is no reflection and the output of the detector will be a few volts.

The disk is installed on the camera mount pad and rotates with it. The signal from the detector is used by the microcontroller to determine a reference position. All the other positions may be calculated relative to this reference. The end of scan position is determined directly from the sensor.

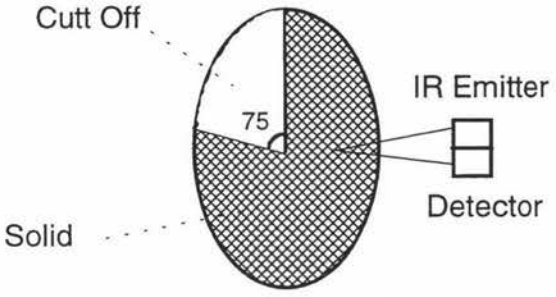


Figure 4- Reflective Disk

Scanner assembly is shown in Figure 5.

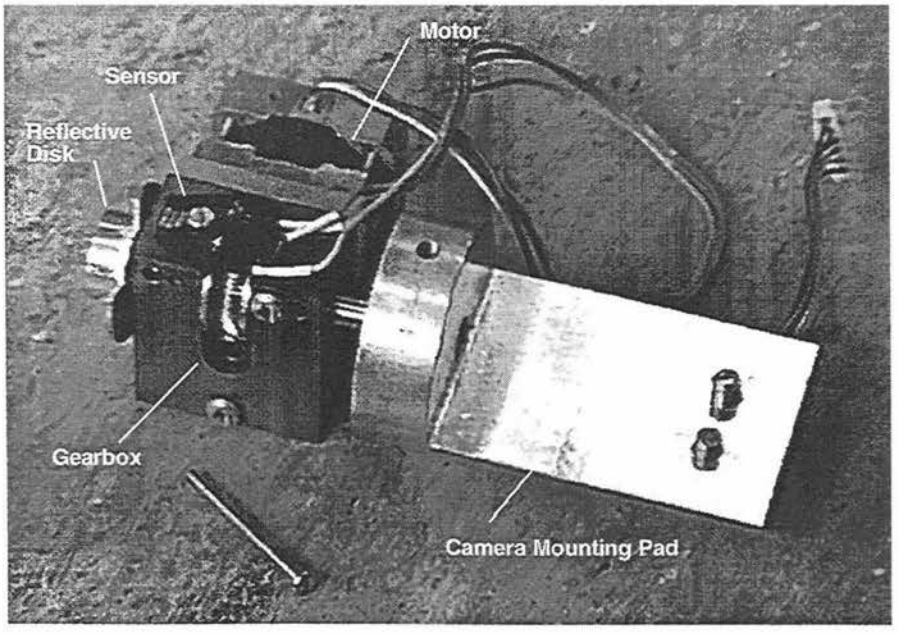


Figure 5- Scanner Assembly

2.2.1.5- Controller Board

The operation of the scanner is controlled by electronics on the controller board. Block diagram of the controller board and its interface to other elements of the system is shown in Figure 6.

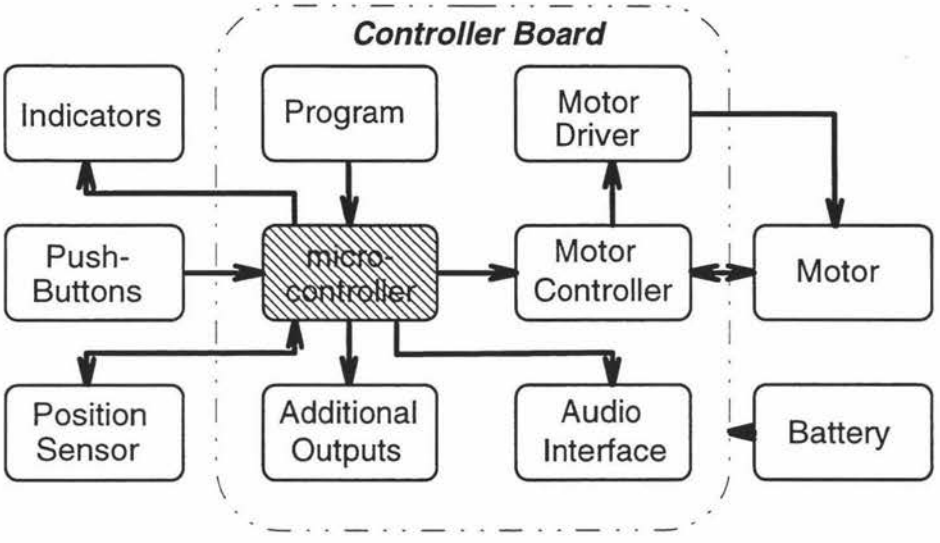


Figure 6- Controller Board Block Diagram

The operation of the unit is controlled by an 8 bit microcontroller. The microcontroller chosen was the popular 80C31 which is made by a number of integrated circuit manufacturers. The program to be executed by the microcontroller is stored external in an EPROM (Electrically Programmable Read Only Memory).

The waveform required to drive the stepper motor is generated by a stepper motor controller chip (L297 manufactured by SGS-Thomson). The speed and direction of movement for the stepper motor is controlled by the microcontroller. The driving current for the stepper motor is provided through a darlington driver chip (UDN2981A).

As it was mentioned earlier valid frames are marked by recording a 1KHz audio signal on them. The microcontroller generates this audio signal. The audio signal generated by the microcontroller must be modified in order to match the audio input characteristics of the camera. The audio signal matching is achieved by the audio interface module. It consists of a voltage divider which attenuates the output from the microcontroller to 10mV. This signal is then AC coupled to the audio input of the camera.

Some additional outputs from the microcontroller are available for future expansion.
The complete circuit diagram for the controller board is shown in Appendix 1.

The controller circuit board and components is shown in figure 7.

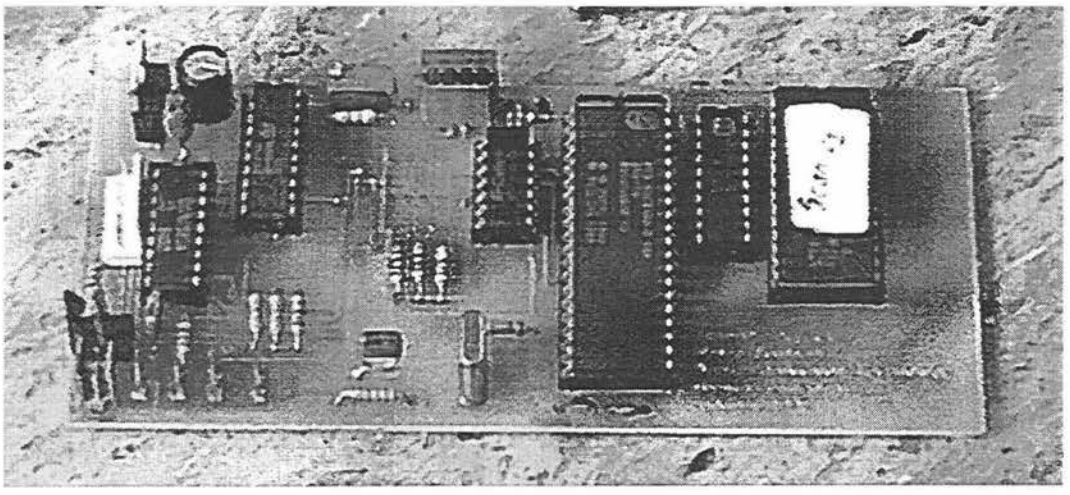


Figure 7- Controller Board

2.2.1.6- Battery

The unit is powered by two internal 6V, 3Ah sealed lead acid batteries. The batteries are connected in parallel to provide a total of 6Ah at 6V. One of the batteries used in the unit is shown in Figure 8. Each battery weights 0.61Kg.

Figure 8- Battery

2.2.1.7- Connections

Electrical connections to the unit consist of a battery charger input for easy charging of the batteries, an audio signal output which is connected to the camera through a cable, and a power output which will be used to power the camera from the unit. (The power output is not implemented yet).

2.2.1.8- Switches and Indicators

The operation of the unit is controlled by 3 push button switches and on/off rocker switch. Push button switches are labelled "Ready", "Scan" and "Stop". The operation of the switches are as following:

- Ready- When "Ready" button is pushed, motor will drive the camera mounting pad to the home position, making the unit ready for scanning.
- Scan- Scanning starts when "Scan" button is pushed,
- Stop- Scanning is terminated when "Stop" button is pushed. "Stop" button is provided to terminate scanning before completion. It is useful when a tree is shorter than 40 meters and does not require the full scan angle to be used. If "Stop" button is not pushed during scanning, full scan will be done. All buttons are illuminated and the light is only on when pushing the button will have an effect. Only 1 light is on at any point in time, providing a very easy user interface of "push the illuminated button" for operation.

The connections, switches and indicators on the unit are shown in Figure 9.

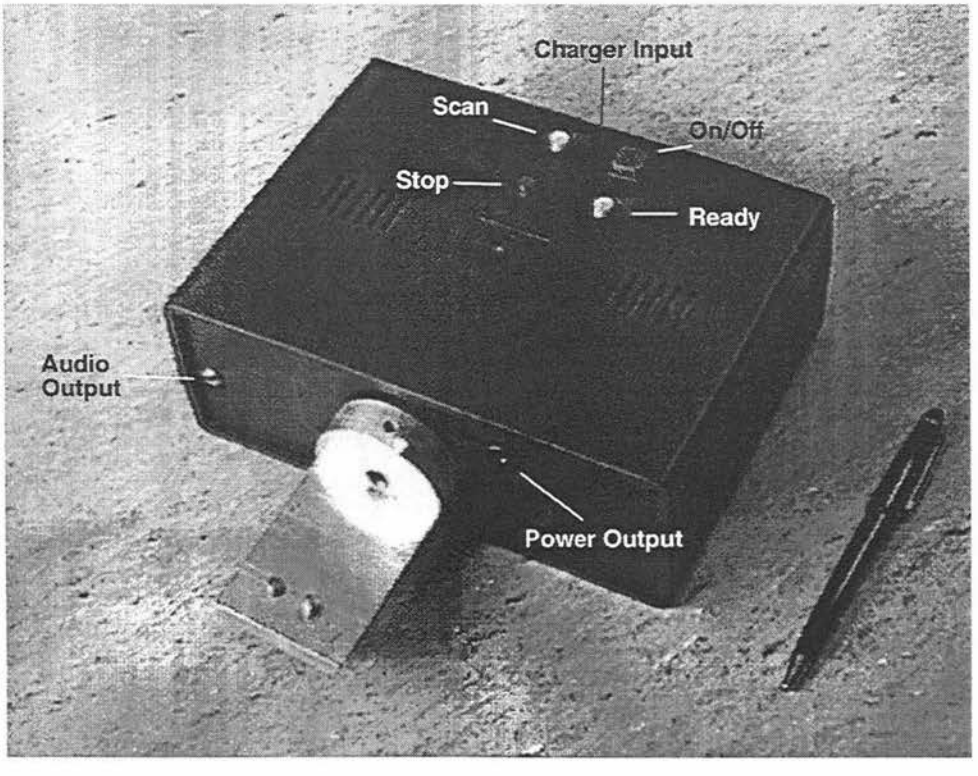


Figure 9- Connections, Switches and Indicators

2.2.1.9- Power Consumption

The power consumption to the unit is managed by the controller board to minimise consumption. The major power consuming element in the unit is the motor. It can draw a maximum of 500mA from a 5V supply. However the design of the circuitry is such that the maximum current drawn by the motor is limited to 400mA. When the motor is turned off, the torque exerted to the motor by the camera, is less than the motor detent torque and therefore it will stay at its position. To minimise the required power, the supply to the motor is turned off by the controller, when it is not moving. During the full scan of a tree, the camera is only moving:

$$400ms \times 21Steps = 8.4Seconds$$

to scan to the top of the tree and 8.4 seconds to go back to the home position, a total movement time of 16.8 seconds per scan. This is equivalent to:

$$16.8Seconds \times 400mA = 6.72ASeconds$$

$$= 1.87mAh \ \ Per\,Scan$$

The rest of the circuitry, draws about 150mA from the batteries for the whole duration that the unit is on.

The power consumption of the unit is therefore:

When motor is moving:

$$(400mA + 150mA) \times 6V = 3.3WattS$$

When motor is not moving: $150mA \times 6V = 0.9Watts$

Assuming that the unit is used for scanning 10 times per hour and is left on all the time, the average power consumption over an hour would be:

Time that motor moves:

$$10Scans \times 16.8Seconds \ / \ Scan = 168Seconds$$

Power consumption while motor moves: $(168/3600) \times 3.3W = 0.15Watt$

Power consumption rest of the time: $((3600 - 168)/3600) \times 0.9W = 0.86Watt$

Average Power Consumption: $0.15 + 0.86 \cong 1Watt$

Average Current input: $1Watt \ / \ 6V = 168mA$

The batteries in the unit have a total capacity of 6Ah . Fully charged batteries will last for 35.7 hours ($6Ah / 0.168mA = 35.7hours$) of continuos operation. Assuming it takes 3 to 4 hours to image a plot, the batteries would last for imaging about 10 plots (ie. about 5 working days!). Excessive capacity of the batteries is available to allow modification to the design in the future such that the camera also would use the power from these batteries. In that case fully charged batteries will last for imaging about 3 plots (ie. over a day of operation of the unit).

2.2.2- Firmware

The program which is used by the microcontroller is stored on a 64KByte EPROM. The source code is written in assembly language for 8051 microcontroller family and since it has only used core resources, may be assembled for any member of the family. The flow diagram of the program is shown in Figure 10. The source code is included in Appendix 2.

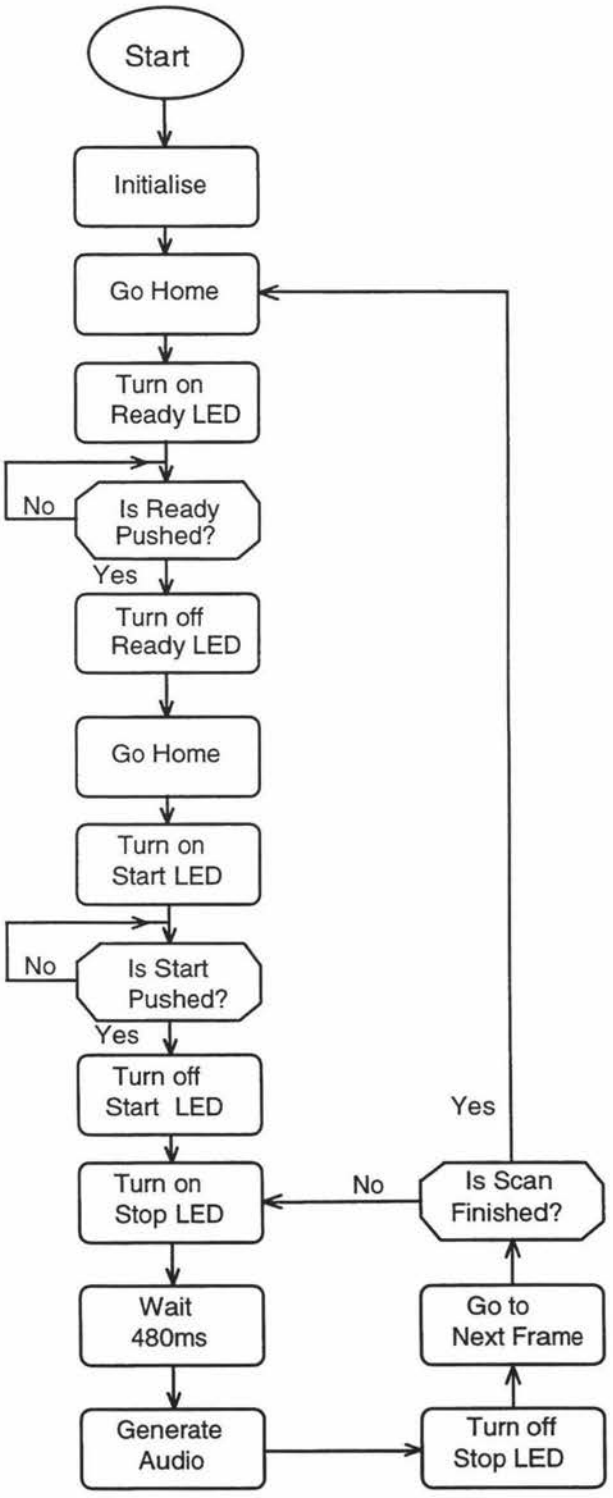


Figure 10- Program Flow Diagram

2.3- Decoder

During the scanning of a tree, many frames are recorded onto the tape of which only few are valid and need to be captured by the frame grabber card. These valid frames are marked by a 1KHz signal recorded on the audio track of the video tape. When the tape is played back, the audio signal is connected to the decoder. The decoder in effect converts the audio signal to a trigger signal which is used by the frame grabber card to capture a valid frame. Figure 11 shows the decoder unit. Circuit diagram for the decoder unit is shown in Appendix 3.

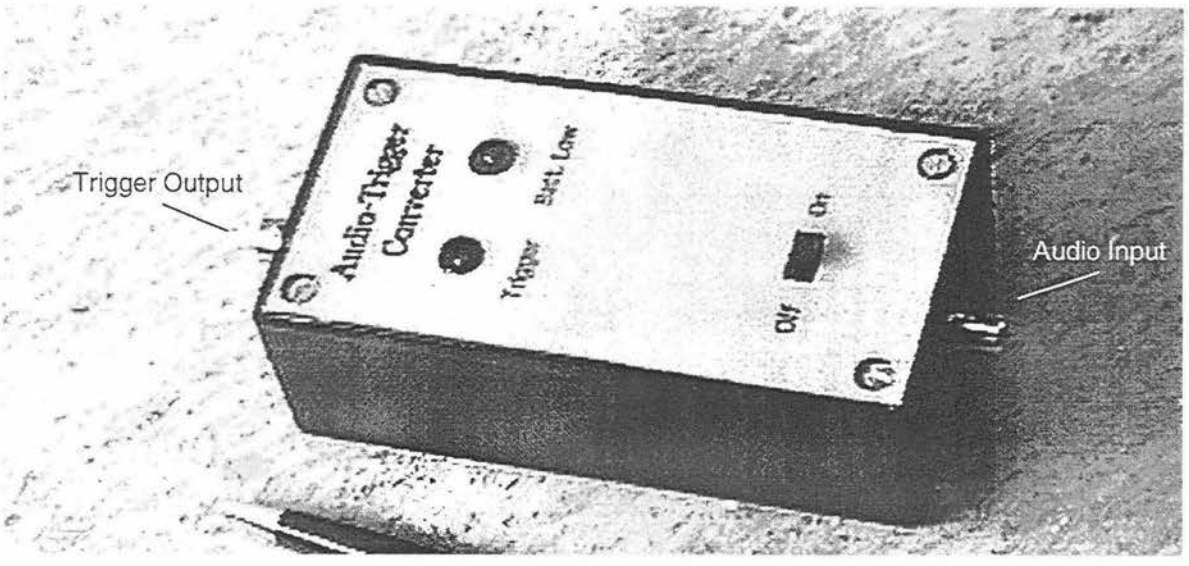


Figure 11- Decoder Unit

2.4- Frame Grabber

The frame grabber (DT2255-50Hz by Data Translation Inc.) is a computer card which is installed into a Macintosh motherboard slot. It is used to capture selected frames from the video tape and store them onto the computer hard disk. It can capture video frames at the rate of 25 frames per second, providing it has enough fast memory on board. For this application capturing an individual frame is prompted by an external trigger signal which is produced by the decoder. The resolution of a captured frame is 768Lines×512Pixels.

2.5- Image Acquisition Software

The frame grabber card needs to be programmed for its operation. The software which is used for controlling the frame grabber card is a public domain application called "NIH-Image". A program was written using the macro instructions of the "NIH-Image" to control the acquisition of the frames. This program instructs the frame grabber card to only capture a frame when an external trigger signal from the decoder is received. It then forms a stack of the individual frames and allows the user to either save or manipulate the stack. The source code of this macro is included Appendix 4.

A sample image acquired using the system is included in Appendix 5.

3- FUTURE WORK

The operation and construction of the system may be improved as following:

3.1- Scanning Platform:

- Gearbox- There is some backlash in the gearbox which adds to the vibration at each stop position. Also there appears to be too much friction between the gears as the output torque is much less than expected. Some work needs to be done to improve the gearbox.

- Platform Mounting- The scanning platform is mounted on the tripod by screwing the standard tripod bolt into the gearbox housing. The gearbox housing is made out of plastic and threads turned into it are not hard enough to withstand the force and get stripped easily. The mounting needs modification.

- Camera Mounting- The axis of rotation of the camera mounting pad must be parallel to the plane of the pad where the camera is positioned. Otherwise a horizontal displacement occurs between consecutive images. Observation of the captured images shows this horizontal displacement. The camera mounting pad needs to be carefully examined and modified to improve this.

- Camera Power- The power for operating the camera is taken from a rechargeable battery connected directly to the camera. This adds to the weight of the camera which increases the load on the motor and also adds to the vibration. The battery charge only lasts for about two hours of the operation of the camera. It is possible to use the internal batteries of the scanning platform to power the camera. This will allow the system to work faster and longer with a full charge. To achieve this the power supply for the system needs to be redesigned.

- Camera Operation- Presently the user is required to manually start the video camera for recording at the start of the scan and stop it when the scan is finished. In addition to the inconvenience, if the user forgets to turn on the camera before scanning or to turn the camera off at the end of scan, there will be a waste of time and tape. Camera operation can be automated through the remote control sensor input of the camera. To achieve this the codes generated by the remote controller unit of the camera needs to be found and generated by the microcontroller.

- Annotating Images- It is possible to annotate the images (eg. tree number, plot number, etc) at the time of capturing on the field, by recording the voice of the user on the audio track. This method will take advantage of the fact that the sound is recorded in stereo. One channel can be used for annotating the whole image, and the other for coding the individual frames. This can be achieved by installing a microphone on the scanning platform, connected to one audio channel of the camera and controlled by the microcontroller so that it is only active at the beginning of each scan.

3.2- Decoder:

- At the time of capturing images into the computer the operator has to start the image acquisition macro, start the video player, stop it after the image has been captured into the computer, save the captured image onto the hard disk of the computer, and repeat the same procedure for every image.

In order to further automate the system, the operation of the decoder may be enhanced by allowing it to receive commands from the computer and instruct the video player accordingly. With this enhancement it may be possible to capture the images into the computer semi automatically. It may still be desirable to name each image file by the operator to reflect the location and other specifics about an image. This data is recorded as voice on the audio track of the tape.

An alternative would be to digitise the voice and store it with the image, such that each tree has an image and a sound file associated with it. This provides a fully automatic data transfer system.

3.3- Image Acquisition Software:

- Image acquisition software could be set to capture a specific number of frames. The number of frames in an image required to cover the full length of a tree may be different depending on the height of the tree. If the number of frames to be captured (set in the software) is less than the actual number of frames, some frames will be missed. If actual frames are more than what is set in the software, then the program will wait indefinitely for the next frames. An improvement is required to rectify this problem.

3.4- Image Construction and Analysis:

3.4.1- A computer program needs to be developed to convert the stack of frames into a complete image of the tree by removing the overlaps and appending the consecutive frames. Followings are possible methods to achieve this:

3.4.1.1- Assuming that the angular movement between consecutive frames in the vertical direction is the same:

i) Find the overlap between the first and the second frame in terms of number of lines by some correlation algorithm;

ii) Remove the obtained number of overlapped lines from the second frame;

iii) Append the first frame and the remaining of the second frame together to form a partially constructed image;

iv) Remove the number of overlapped lines from the next frame;

v) Append the partially constructed image and the remaining of the next frame together to form a new partially constructed image;

vi) Repeat steps (iv) and (v) until all frames have been processed. The resultant image will be a fully constructed image of the tree.

3.4.1.2- Make no assumptions about the angular movement between consecutive frames in the vertical direction:

Same as 3.4.1.1 except have to perform correlation algorithm for all frames, rather than only first and second.

This method will take longer processing time.

3.4.1.3- Assume that the angular movement between consecutive frames in the vertical direction has an accuracy of say ±10%:

i) Find the overlap between the first and the second frame in terms of number of lines by some correlation algorithm;

ii) Remove the obtained number of overlapped lines from the second frame;

iii) Append the first frame and the remaining of the second frame together to form a partially constructed image;

iv) Remove say 90% of the number of overlapped lines from the next frame to form a partially processed frame;

v) Find the overlap between the partially processed frame and the partially constructed image in terms of number of the lines by some correlation algorithm;

vi) Remove the obtained number of overlapped lines from the partially processed frame;

vii) Append the partially constructed image and the remaining of the partially processed frame together to form a new partially constructed image;

viii) Repeat steps (iv) to (vii) until all frames have been processed.

The resultant image will be a fully constructed image of the tree. This method is a combination of methods 3.4.1.1 and 3.4.1.2 and is slower than 3.4.1.1 and faster than 3.4.1.2.

All the above mentioned methods assume that there is no horizontal movement between frames. Horizontal movement could occur due to mechanical instability or misalignment of the camera sensor and the camera mount pad. If this happens, then a two dimensional correlation algorithm must be used which requires a much longer processing time to construct an image.

3.4.2- Once an image is formed, an analysis needs to be done to remove distortions introduced by imperfections in the lens, movement of the camera, perspective distortion, etc. The resulting image must be suitable for accurate measurements.

Image construction and analysis seems to be the most challenging part of the work to be undertaken.

4- SUMMARY

A prototype high resolution videotape based tree image system was developed. The operation of the system is as following:

- A video tape camera is mounted on a scanner which is fixed on a tripod.
- While the camera is recording, scanner tilts along the stem of the tree in steps of equal angle.
- At each step, scanner stops for a short time and the frames captured at that time are audibly marked as valid frames.
- To capture the valid frames into a computer, the tape is then played back and the video signal is connected to a frame grabber card installed into a computer.
- The audio signal is played through a decoder which generates a trigger signal for the frame grabber to capture only the valid frames. Frames are sequential with some overlap between the adjacent frames.
- Once all the valid frames are captured, a computer program will put individual frames together, forming a complete image of the object (This is phase 2 of the implementation and is yet to be done).

5- ACKNOWLEDGMENTS

- I would like to thank all the people who helped me in one way or another in developing this system. In particular I would like to thank Professor Bob Hodgson, Head of the Department of Production Technology and Mr. Ralph Pugmire for providing encouragement, support and direction in this development.

- I would also like to thank Mr. Gary Allen for his involvement in the design and assembly of the controller board and Mr. Thomas Look for making the mechanical parts.

- Financial support from Tasman Forestry is greatly appreciated.

6- REFERENCES

Pugmire, Ralph, *Automation of Forest Stand Assessment Feasibility Study for Tasman Forestry,* 1993, Department of Production technology, Massey University.

Jack, Keith,1955, *Video demystified: A hand book for the digital engineer*, HighText Publications, Inc. ISBN 1-878707-09-4

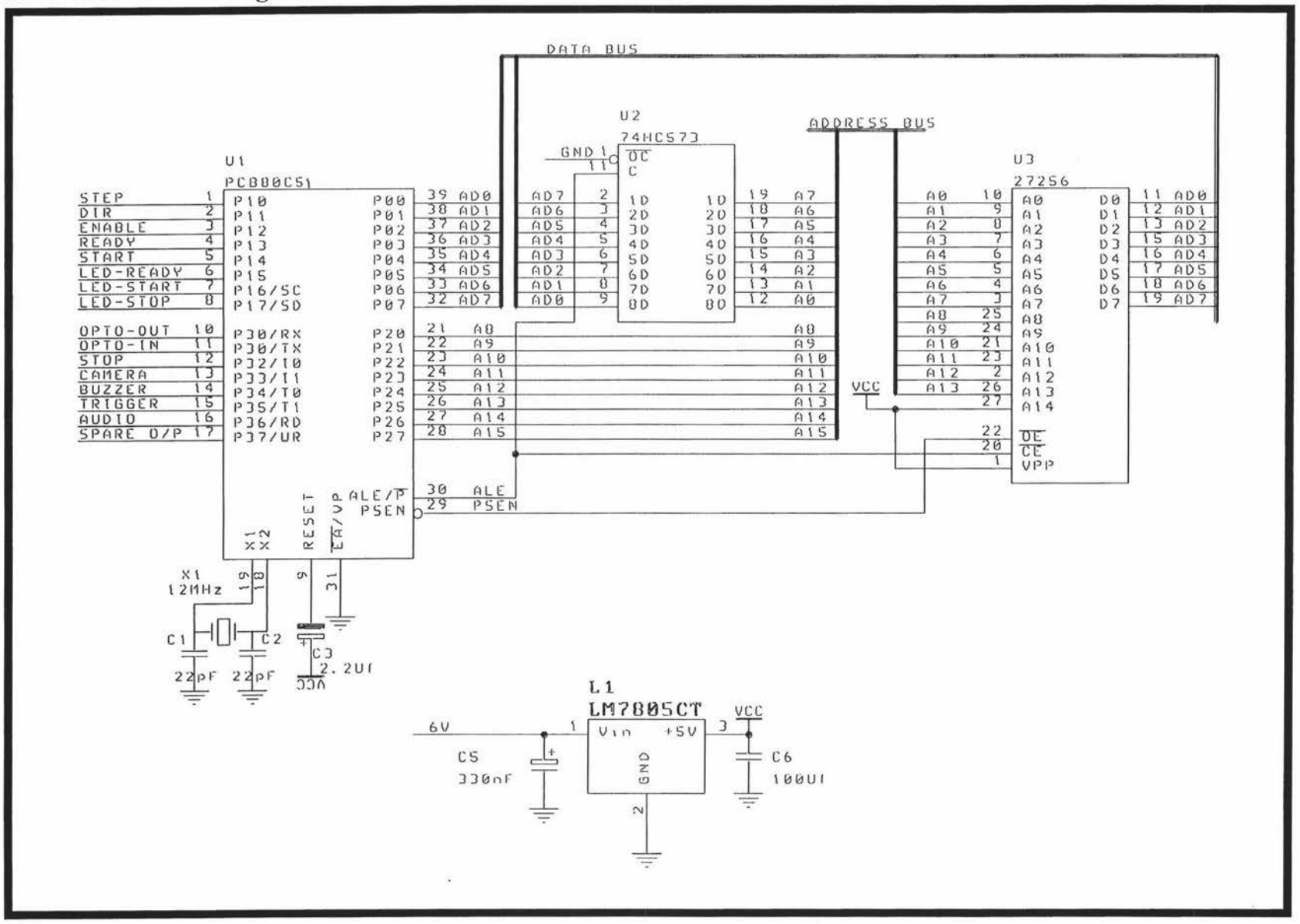
AKAI Electronic Co., Ltd. 1992, *AKAI Video Camera Recorder Operators Manual*, AKAI Electronic Co., Ltd.

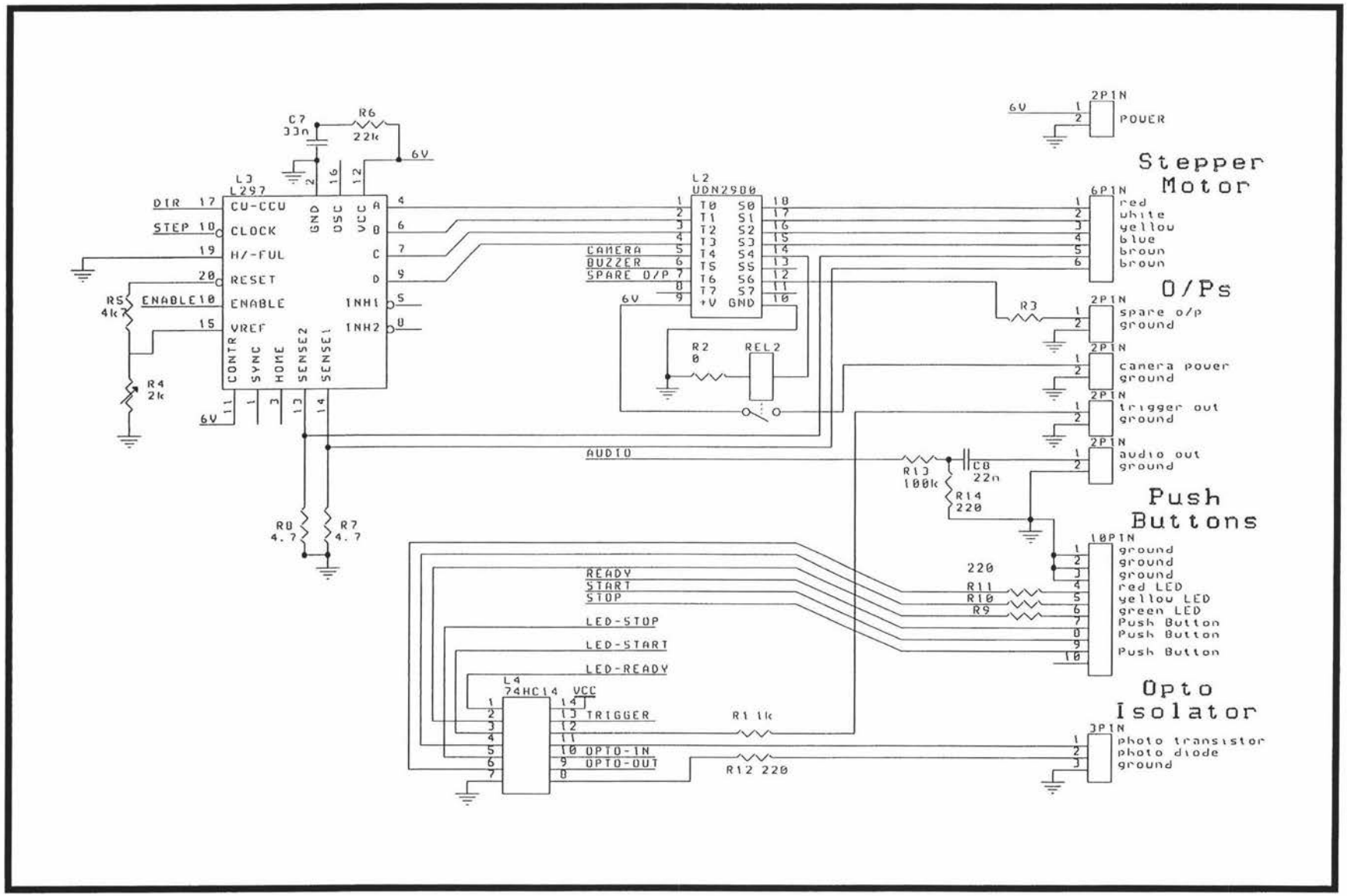
Technical Support Department, 1988, *Quick Capture User Manual,* Data Translation, Inc.

Rasband, Wayne, 1994, *NIH-Image User Manual.*

Components Data Sheets by various manufacturers .

Controller Board Circuit Diagram




L3
L297
DIR 17
STEP 10
19
20
ENABLE 18
15
CW-CCW
CLOCK
H/-FUL
RESET
ENABLE
VREF
CONTR
SYNC
HOME
SENSE2
SENSE1
GND
OSC
VCC
A
B
C
D
INH1
INH2
C7
33n
R6
22k
6V
R5
4k7
R4
2k
R8
4.7
R7
4.7
L2
UDN2900
CAMERA
BUZZER
SPARE O/P
R2
0
REL2
AUDIO
R13
100k
C8
22n
R14
220
2PIN
POWER
Stepper Motor
6PIN
red
white
yellow
blue
brown
brown
O/Ps
R3
spare o/p
ground
camera power
ground
trigger out
ground
audio out
ground
Push Buttons
10PIN
ground
ground
ground
red LED
yellow LED
green LED
Push Button
Push Button
Push Button
220
R11
R10
R9
READY
START
STOP
LED-STOP
LED-START
LED-READY
L4
74HC14
VCC
TRIGGER
OPTO-IN
OPTO-OUT
R1 1k
R12 220
Opto Isolator
3PIN
photo transistor
photo diode
ground

Program Source Code

```
;*****************************************************************
;     Hardware definitions for motor interface
;*****************************************************************
          Motor_Step              equ     P1.0
          Motor_Dir               equ     p1.1
          Motor_Enable            equ     p1.2

;*****************************************************************
;     Hardware definitions for push buttons interface
;*****************************************************************
         Pb_Ready                 equ     p1.3
         Pb_Start                 equ     p1.4
         Pb_Stop                  equ     p3.2

;*****************************************************************
;     Hardware definitions for push button LEDs interface
;*****************************************************************
         Pb_Ready_LED             equ     p1.5
         Pb_Start_LED             equ     p1.6
         Pb_Stop_LED              equ     p1.7

;*****************************************************************
;     Hardware definitions for position sensors interface
;*****************************************************************
          Pos_Sens_Emitter        equ     p3.0
         Pos_Sens                 equ     p3.1


;*****************************************************************
;     Other definitions
;*****************************************************************
         Camera_Relay             equ     p3.3
          Buzzer                  equ     p3.4
          Trigger                 equ     p3.5
         Audio                    equ     p3.6

          no_steps                equ     R0
          stop_flag               equ     00h

;*****************************************************************
;                          Main Program
;*****************************************************************
                  org     0000h                   ;Reset routine
resethere:        jmp     start

                  org     resethere+3             ;INT0 routine
                  ljmp    pb_stop_int

start:            mov     sp,#5Fh
                  lcall   initialise
                  lcall   scan
                  ret
```

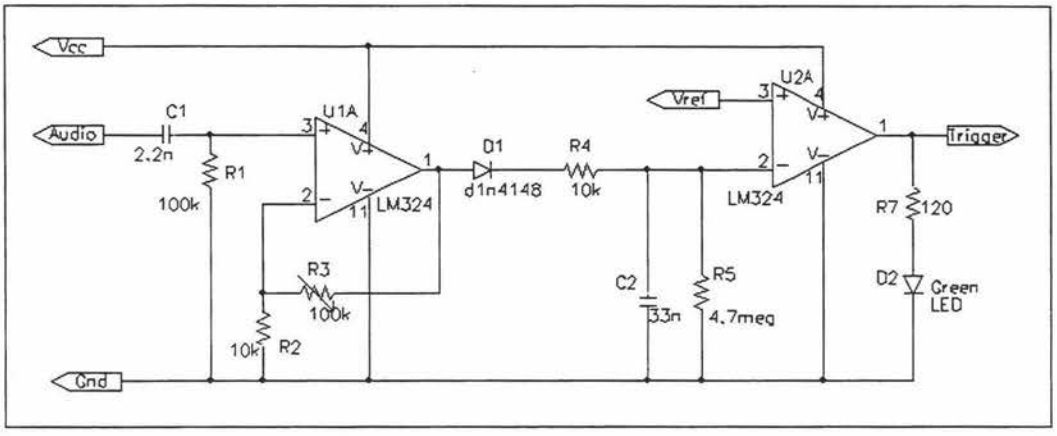
```
initialise:
                mov     p1,#00011010b       ;initialise port 1
                mov     p3,#10000110b       ;initialise port 2
                mov     ie,#10000001b       ;enable external interupt 0
                mov     no_steps,#200       ;set twice the number of steps
                                            ;to go from one frame to the next
                lcall   Go_Home             ;go to home position
                ret


scan:
                setb    pb_Ready_Led        ;turn off LED's
                setb    pb_Start_Led
                setb    pb_Stop_Led

Wait_Ready:     clr     stop_flag           ;clear stop flag
                clr     pb_Ready_Led        ;turn on ready LED
                jb      Pb_Ready,Wait_Ready ;Wait for the Ready push button
                setb    pb_Ready_Led        ;turn off ready LED

Camera_On:      setb    Camera_Relay        ;turn on the camera
                lcall   Go_Home             ;go to home position
                lcall   Wait_240ms          ;wait for the camera set up
                clr     pb_Start_Led        ;turn on Start LED

Wait_Start:     jb      Pb_Start,Wait_Start ;Wait for Start push button
                setb    pb_Start_Led        ;turn off start LED
                clr     Pos_Sens_Emitter    ;turn on position sensor emitter

Scan_Loop:      clr     pb_Stop_Led         ;turn on Stop LED
                lcall   Wait_240ms          ;wait 240ms for camera to stabilise
                lcall   Wait_240ms          ;extra times for settlement
                lcall   Gen_Audio           ;generate audio signal
                setb    pb_Stop_Led         ;turn off stop LED
                lcall   Go_Up               ;go to the next frame position

                lcall   Wait_100ms
                jb      Pos_Sens,Scan_Done  ;if at high position, turn camera off
                ljmp    Scan_Loop           ;repeat until scan is complete

Scan_Done:      clr     Camera_Relay        ;turn off the camera
                setb    Buzzer              ;Sound the buzzer for one second
                lcall   Wait_240ms
                lcall   Wait_240ms
                clr     Buzzer
                lcall   Go_Home
                setb    Pos_Sens_Emitter
                ljmp    Wait_ready


Go_Home:        setb    Motor_Enable        ;enable motor
                jb      stop_flag,scan      ;goto ready if stop been pushed go
                clr     Pos_Sens_Emitter    ;Turn on Position sensor
                lcall   wait_240ms          ;give time for sensor to
                lcall   wait_240ms          ;to settle
                jnb     Pos_Sens,Go_Down    ;go down if within range
                mov     A,#2
  loop_home:    lcall   Go_Up               ;go up two frames
```

```
                dec     A
                jnz     loop_Home
                jnb     Pos_Sens,Go_Down        ;go down if within range
                lcall   Find_Range              ;else find range
                lcall   Go_Down                 ;now go down
                setb    Pos_Sens_Emitter        ;Turn off Position Sensor
                clr     Motor_Enable            ;Disable motor
                ret


Find_Range:     setb    Motor_Enable            ;Enable motor
                clr     Motor_Dir               ;go down
Find_Loop:      jb      stop_flag,scan          ;goto ready if stop pushed
                cpl     Motor_Step
                lcall   Wait_2ms
                jb      Pos_Sens,Find_Loop      ;until within range
                clr     Motor_Enable            ;Disable motor
                ret

;go down until at lowest position
Go_Down:        setb    Motor_Enable            ;Enable motor
                clr     Pb_Stop_Led             ;turn on stop LED
                clr     Motor_Dir               ;go down
        L1:     jnb     stop_flag,continue1
                jmp     scan                    ;goto Ready if stop pushed
continue1:      cpl     Motor_Step
                lcall   Wait_2ms
                jnb     Pos_Sens,L1             ;until out of range
                setb    Motor_Dir               ;go up
        L2:     jnb     stop_flag,continue2
                jmp     scan                    ;goto scan if stop pushed
continue2:      cpl     Motor_Step              ;
                lcall   Wait_2ms
                jb      Pos_Sens,L2             ;until within range
                Setb    pb_Stop_Led             ;turn off stop LED
                clr     Motor_Enable            ;Disable motor
                ret

;go to the naxt frame position.
Go_Up:          setb    Motor_Enable            ;Enable motor
                setb    Motor_Dir               ;go up
                mov     R1,no_steps             ;load twice the no of steps
        L3:     jnb     stop_flag,continue
                jmp     scan                    ;goto ready if stop pushed
 continue:      cpl     Motor_Step
                lcall   Wait_2ms                ;step frequency=250
                djnz    R1,L3                   ;step for the no of steps
                clr     Motor_Enable            ;Disable motor
                ret


;Generate audio signal for 240ms
Gen_Audio:
                mov     R6,#240                 ;load 240ms
                mov     A,tmod
                anl      A,#0f0h            ;mask off high nibble of TMOD
```

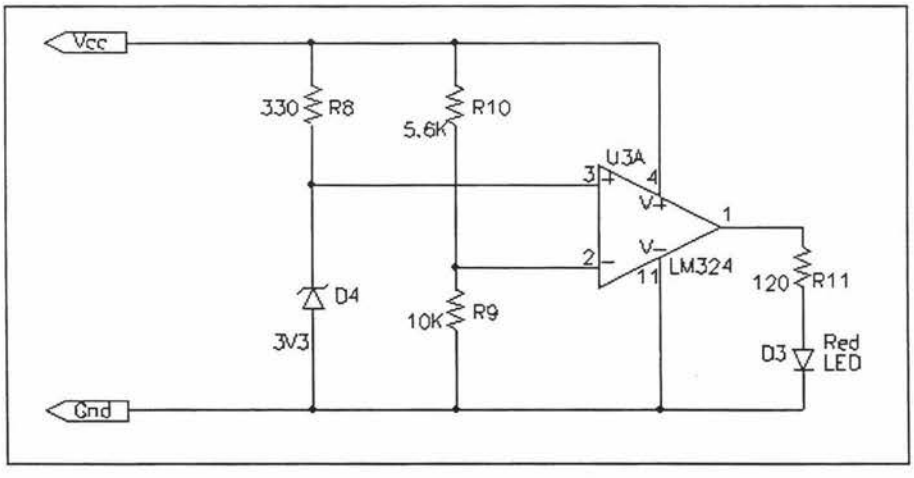
```
                orl     A,#01h          ;set timer 0 at mode 1
                mov     tmod,A          ;ie. 16 bit timer/counter
                clr     Trigger
    loop:       clr     tr0             ;turn off timer 0
                mov     tl0,#00Bh       ;load timer with FE0Bh (FFFFh-01F4h)
                mov     th0,#0FEh       ;01F4h=500us/1us (fosc=12Mhz)
                setb    tr0             ;turn on timer 0
                clr     tf0             ;clear timer overflow flag
    wait:       jnb     tf0,wait        ;wait for the overflow flag
                cpl     Audio           ;toggle signal
                djnz    R6,loop         ;until 240ms is over
                clr     tr0             ;turn off timer 0
                setb    Trigger
                ret

;Stop Button Intrrupt Routine
 pb_stop_int:
                push    Acc             ;save accumulator
                push    PSW             ;save PSW
                mov     psw,#08         ;clear all falgs and use register bank 1
                setb    Buzzer          ;Sound the buzzer
                lcall   Wait_100ms      ;Wait for the buzzer
                lcall   Wait_240ms
                clr     Buzzer          ;Stop buzzer
                clr     pb_Ready_LED    ;turn on ready LED
                setb    pb_Start_LED    ;turn off start LED
                clr     pb_Stop_LED     ;turn on Stop LED
                setb    stop_flag       ;set stop flag
                clr     Camera_Relay    ;turn off camera
                pop     psw             ;restore staus
                pop     Acc
                reti

;*****************************************************************************
;       Dly_2ms provides 2ms wait time
;*****************************************************************************
Dly_2ms:
                mov     R3,#3                   ;2ms = 3 X 222 X (3 X 1us)
                xch     A,R4                    ;save ACC
    out_loop:   mov     A,#222
    in_loop:    dec     A                       ;each loop take 3 machine cycles
                jnz     in_loop                 ;each machine cycle is 1us with 12Mhz
clock
                djnz    R3,out_loop
                xch     A,R4                    ;restore ACC
                djnz    R5,Dly_2ms
                ret


Wait_2ms:
                mov   R5,#1
                lcall Dly_2ms
                ret
```

```
;*************************************************************************
;     Wait_240ms provides 240ms wait time
;*************************************************************************
Wait_240ms:

                mov     R5,#120
                lcall   Dly_2ms
                ret
;*************************************************************************
;     Wait_100ms provides 100ms wait time
;*************************************************************************
Wait_100ms:
                mov   R5,#50
                lcall Dly_2ms
                ret
```

Decoder Circuit Diagram



Trigger Generator

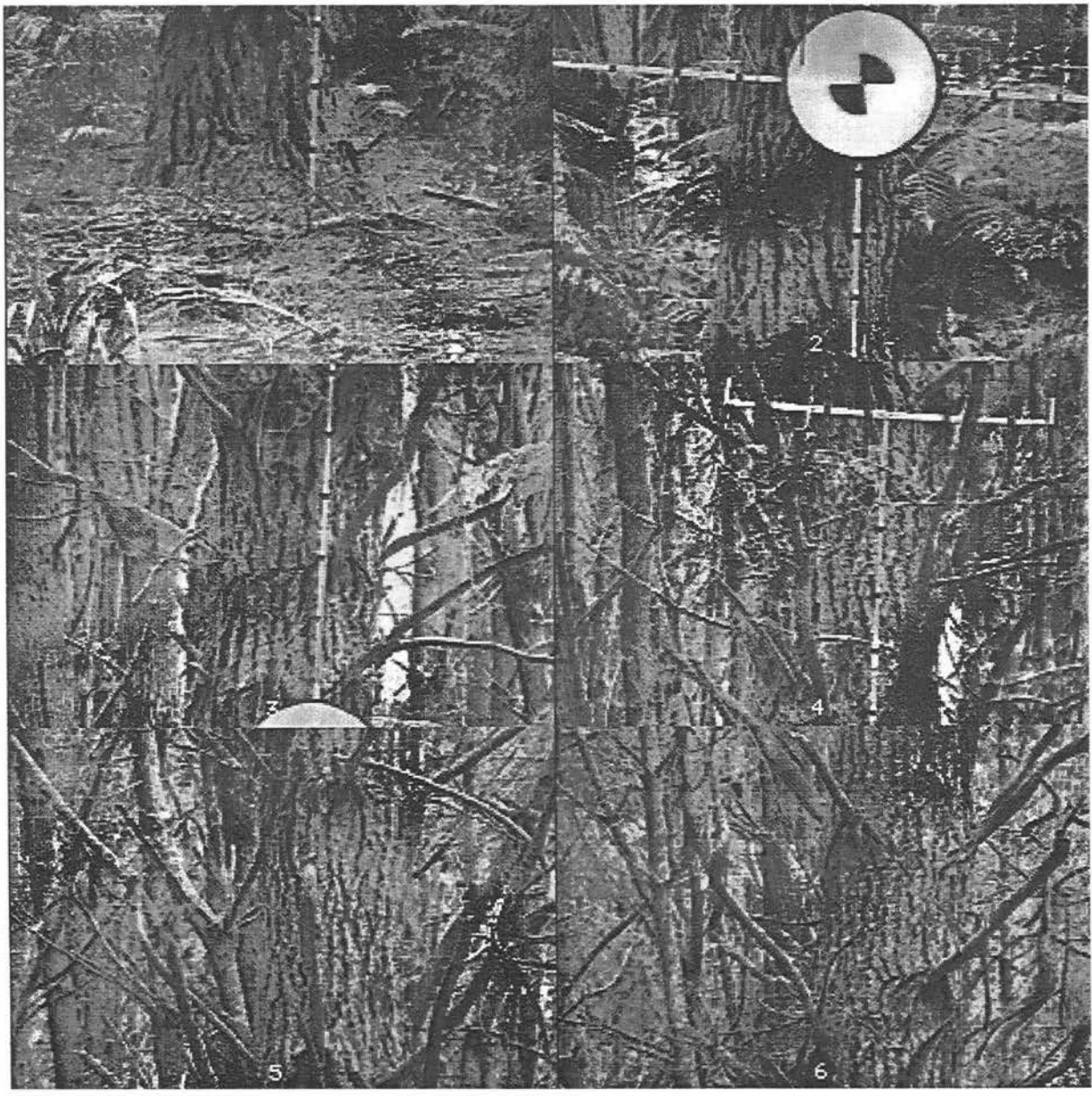


Low Battery Voltage Detection

Image Acquisition macro

```
Macro 'Acquire_ [F12]';
var
 nFrames,i:integer
 pid, x,y,width,height:integer;
begin
        nFrames:=20;
        width:=768;
        height:=512;
        SaveState;
        SetNewSize(width,height);
        MakeNewStack('scan');
        WaitForTrigger;
        for i:=1 to nFrames do begin
                StartCapturing;
                WaitForTrigger;
                selectall;
                copy;
                selectwindow('scan');
                selectslice(i);
                paste;
                if i<>nFrames then AddSlice;
        end;
        RestoreState;
end;
```

Sample Image




7
8
9
10
11
12


13
14
15
16
17
18

APPENDIX B- MACRO SOURCE CODE

```
var
        PrefsPath,buildPath:string;fftsize:integer;
        Hshift,Voverlap:integer;

procedure CheckForSelection;
var
        x1,y1,x2,y2,LineWidth:integer;
begin
        GetRoi(RoiLeft,RoiTop,RoiWidth,RoiHeight);
        GetLine(x1,y1,x2,y2,LineWidth);
        if (RoiWidth=0) or (x1>=0) then
        begin
                PutMessage('Please make a rectangular selection.');
                exit;
        end;
end;

procedure CheckForStack;
begin
        if nPics=0 then
        begin
                PutMessage('This macro requires a stack.');
                exit;
        end;
        if nSlices=0 then
        begin
                PutMessage('This window is not a stack.');
                exit
        end;
end;

procedure fftMatchProc;
var
        left,top:integer;
        pic1,pic2:integer;width,height:integer;cross:integer;
        tempPID:integer;woswasiiPID,localPID:integer;

begin
        PrefsPath:=GetPath('pref');
        PrefsPath:=concat(PrefsPath,'AM Prefs Ä:');
        BuildPath:=concat(PrefsPath,'Mean(7x7)');
        {                fftsize:=256;}
        chooseSlice(2);
        stack:=PicNumber;
        GetPicSize(width,Height);
        left:=(width-(fftsize-1))/2;
        top:=height-(fftsize-1);
        MakeRoi(left,top,fftsize-2,fftsize-2);
        ResetCounter;
        Measure;
```

```
{addconstant(-rMean[1]);}
SetBackground(round(rMean[1]));
Copy;
SetNewSize(FFTsize,FFTsize);
MakeNewWindow('Temp');
MakeRoi(1,1,FFTsize-1,FFTsize-1);
Paste;
KillROI;
woswasiiPID:=PIDnumber;
Duplicate('temp1');
Convolve(BuildPath);
localPID:=PidNumber;
ChoosePIc(woswasiiPID);
Filter('smooth');
ImageMath('sub',woswasiiPID,LocalPID,20,50,LocalPID);
fft('foreward');
pic2:=PIDnumber;
ChoosePic(woswasiiPID);
Dispose;
ChoosePic(LocalPID);
Dispose;
selectPic(stack);
chooseSlice(1);
Top:=Voverlap - FFTSize
If Top < 0 then Top:=0;
MakeRoi(left,Top,fftsize,fftsize);    {top changed from 0 to Top}
ResetCounter;
Measure;
{addconstant(-rMean[1]);}
SetBackground(round(rMean[1]));
Copy;
SetNewSize(FFTsize,FFTsize);
MakeNewWindow('Temp');
MakeRoi(1,1,FFTsize-1,FFTsize-1);
Paste;
KillROI;
woswasiiPID:=PIDnumber;
Duplicate('temp1');
Convolve(BuildPath);
localPID:=PidNumber;
ChoosePIc(woswasiiPID);
Filter('smooth');
ImageMath('sub',woswasiiPID,LocalPID,20,50,LocalPID);
fft('foreward');
pic1:=PIDnumber;
ChoosePic(woswasiiPID);
Dispose;
ChoosePic(LocalPID);
Dispose;
imagemath('cmul',pic1,pic2,1,0,'cross');
fft('inverse');
cross:=PIDnumber;
selectpic(pic1);
dispose;
selectpic(pic2);
dispose;
selectpic(cross);
{duplicate('temp');}
```

```
        ResetCounter;
        Measure;
        SetThreshold(rMax[1]);
        MakeBinary;
        SetScale(1,'pixel');
        AnalyzeParticles('include,reset');
        Dispose;
        {putmessage('rX[1]=', rX[1]);
        putmessage('rY[1]=', rY[1]);}
        if rX[1] > fftsize /2 then rX[1]:=rX[1]-fftsize;
                {correct for negative horizontal shift}
        if rY[1] < fftsize /2  then  rY[1]:=rY[1] + fftsize;
                        {correct for vertical shift greater than fftsize}
        rUser1[2]:=round(rX[1]);
        rUser1[1]:=round(rY[1]);
end;

Macro 'appendmacro[z]'
var
        i,j,next,stack,noval:integer;
        row,width,height:integer;
        stop,yleft,NonOverlap,start:integer;
        slices,image,NewHeight:integer;
        Horshift,sliceno,widthshift:integer;
        Strip,FeatureBlockRow,FeatureBlockColumn,MeasureReport:integer;
        overlapstr:string;
        xleft,NewWidth:integer;
        Vstart,Vrange,Vend,Hstart,Hend,Hrange,HrangeStart:integer;
        range,dummy:integer;
        BlockCheck:Boolean;
        TimeTaken:integer;
        StartTime:integer;
        oldheight:integer;
        blind,ExecOnce:boolean;
        fftMatch:boolean;
        VerVariance,HorVariance:integer;
        VerSum,HorSum:integer;
        HorMean,VerMean:integer;
        VerSumSqr,HorSumSqr:integer;
        MaxVerVariance,MaxHorVariance:integer;
        VerReliable ,HorReliable:boolean;
        TempPID,SlicePID:integer;

begin
        fftMatch:=true;
        blind:=false;
        blockcheck:=false;
        dummy:=0;
        range:=0;
        MeasureReport:=0;                       {0 for no report,  1 for report}
        Strip:=1;
        FeatureBlockRow:=30;
        FeatureBlockColumn:=30;
        Vstart:=0;
        Vend:=300;
        Vrange:=50;
        Voverlap:=0;            {added with change to top in FFT proc}
        HrangeStart:=20;
```

```
Hrange:=10;
ExecOnce:=false;
start:=1;
VerSum:=0;
HorSum:=0;
VerSumSqr:=0;
HorSumSqr:=0;
VerMean:=0;
HorMean:=0;
VerVariance:=0;
HorVariance:=0;
MaxVerVariance:=0;
MaxHorVariance:=0;
SaveState;
CheckForStack;
slices:=nslices;
GetPicSize(width,height);
stack :=picnumber;
resetCounter;
startTime:=TickCount;
rUser1[3]:=Vstart;
rUser1[4]:=Vend;
rUser1[5]:=0;
                rUser1[6]:=HrangeStart;
Horshift:=0;
sliceno:=0;

if fftMatch then
begin
        Usercode('append',strip,dummy,dummy);
                Voverlap:=rUser1[1];
        if Voverlap < 40 then fftsize:=32
        else if  Voverlap < 80 then fftsize:=64
        else if Voverlap < 160 then fftsize:=128
        else fftsize:=256;
        putmessage('overlap=',Voverlap);
        putmessage('fftsize=',fftsize);
end;

for i:=1 to nslices do
begin
        chooseSlice(1);
        {traceedges;}
        duplicate('temp');
        {TempID:=PicNumber;}
        saveas(concat('MacintoshHD:Temp:Temp00',i));
        dispose;
        choosepic(stack);
        deleteSlice;
end;

for i:=1 to 50 do
begin
        rUser1[i]:=0;
        rUser2[i]:=0;
end;

for i:=start to slices-1  do
```

```
begin
                        {       resetcounter;}
        if i=start then
         begin
                rUser1[3]:=Vstart;
                rUser1[4]:=Vend;
                rUser1[5]:=0;
                rUser1[6]:=HrangeStart;
                Horshift:=0;
                sliceno:=0;
                NewHeight:=slices*Height/1.11;
                        {minimum 10% overlap between concecutive frames}
                NewWidth:=Width+Hrange;
                SetNewSize(NewWidth,NewHeight);
                MakeNewWindow('Image');
                image:=PidNumber;
                {ChoosePic(stack);
                ChooseSlice(start);}
                open(concat('Macintosh HD:Temp:Temp00',i));
                SelectAll;
                copy;
                dispose;
                ChoosePic(stack);
                addslice;
                paste;
                chooseslice(1);
                deleteslice;
                choosePic(image);
                yleft:=NewHeight-Height;
                xleft:=Hrange / 2;
                MakeRoi(xleft,yleft,Width,Height);
                Paste;
        end;

        if  blind then
        begin
                rUser1[1]:=MeasureReport;
                if  MeasureReport=1 then
                begin
                        SetUser1Label('Vshift');
                        SetUser2Label('Hshift');
                        UpdateResults;
                end;
        end;

        open(concat('Macintosh HD:Temp:Temp00',i+1));
        SelectAll;
        copy;
        dispose;
        choosePic(stack);
        addslice;
        paste;
        if not ExecOnce then
        begin
                if fftMatch then fftMatchProc
                else
                begin
                        if Blockcheck then
```

```
Usercode('BlockMatch',range,FeatureBlockcolumn,FeatureBlockrow);
                        if not BlockCheck then
                                Usercode('StripMatch',strip,dummy,dummy);
                end;
        end;

        if blind then
        begin
                blind:=false;
                ExecOnce:=true;
        end;

        Voverlap:=rUser1[1];
        Hshift:=rUser1[2];
        SetUser1Label('Vshift');
        SetUser2Label('Hshift');
        rUser1[10+i]:=Voverlap;
        rUser2[10+i]:=Hshift;
        {               putmessage('Voverlap=', Voverlap);
        putmessage('Hshift=', Hshift);}
        if blockCheck then
                Vstart:=Voverlap-Vrange/2-FeatureBlockRow;
        if not blockcheck then
                Vstart:=Voverlap-Vrange/2;
        if Vstart< 0 then
                Vstart:=0;
        Vend:=Vstart+Vrange;
                {define the verticl search range}
        if BlockCheck then
                if Vend > (Height-FeatureBlockRow-1) then
                        Vend:=(Height -FeatureBlockRow-1);
        if not blockcheck then
                 if Vend > (Height-Strip-1) then
                        Vend:=(Height -strip-1);
        rUser1[3]:=Vstart;
        rUser1[4]:=Vend;
        {putmessage('Vstart=', Vstart);
        putmessage('Vend=', Vend);}
        {Hstart:=Hshift-Hrange/2;
        Hend:=Hshift+Hrange/2-m;
        if Hstart<0 then Hstart:=0;}
        rUser1[5]:=Hshift;
        rUser1[6]:=Hrange;
        choosePic(stack);
        ChooseSlice(start);
        DeleteSlice;
        chooseSlice(start);
        NonOverlap:=Height-Voverlap;
        MakeRoi(0,0,Width,NonOverlap);
        copy;
        KillRoi;
        ChoosePic(image);
        yleft:=Yleft-NonOverlap;
        Horshift:=Horshift+Hshift;
        widthshift:=Width-Horshift;
        MakeRoi(Horshift+xleft,yleft,width,NonOverlap);
        paste;
```

```
        if i=trunc((slices/2)) then
        begin
                ChoosePic(image);
                selectall;
                copy;
                dispose;
                oldheight:=NewHeight;
                NewHeight:=slices*Height/1.11;
                        {minimum 10% overlap between concecutive frames}
                NewWidth:=Width+Hrange;
                SetNewSize(NewWidth,NewHeight);
                MakeNewWindow('Image');
                image:=PidNumber;
                makeroi(0,Newheight-Oldheight,newwidth,oldheight);
                yleft:=yleft+newheight-oldheight;
                paste;
        end;
end;

KillRoi;
choosePic(stack);
ChooseSlice(start);
DeleteSlice;
Dispose;
{selectwindow('camera');
Dispose;}
{SetExport('measurements');
Export('farshad');}
ChoosePic(image);
for i:=1 to slices do
begin
        rUser1[i]:=rUser1[10+i];
        rUser2[i]:=rUser2[10+i];
        VerSum:=VerSum + rUser1[i];
        HorSum:=HorSum + rUser2[i];
        VerSumSqr:=verSumSqr + (rUser1[i]*rUser1[i]);
        HorSumSqr:=HorSumSqr + (rUser2[i] * rUser2[i]);
end;

VerMean:=VerSum / (slices -1);
HorMean:=HorSum / (slices -1);
putmessage('VerMean = ',VerMean);
putmessage('HorMean = ',HorMean);
putmessage('VerSumSqr = ',VerSumSqr);
putmessage('HorSumSqr= ',HorSumSqr);
VerVariance:=(VerSumSqr / (slices -1)) - (VerMean * VerMean);
HorVariance:=(HorSumSqr / (slices -1)) - (HorMean * HorMean);
for i:=1 to slices do
begin
        if ((rUser1[i] - Mean) * (rUser1[i] - Mean)) > (VerVariance * (error
                                ^2)) then
                putmessage('error is possible');
                { The aim here is to single out one matching which may be quite
                        different from others and flag a unreliable match}
end;
if VerVariance > MaxVerVariance then
        Verreliable:=false;
if HorVariance > MaxHorVariance then
```

```
                Horreliable:=false;
        makeroi(0,yleft,newwidth,newheight-yleft);
        copy;
        dispose;
        oldheight:=NewHeight;
        NewHeight:=oldheight-yleft;
        NewWidth:=Width+Hrange;
        SetNewSize(NewWidth,NewHeight);
        MakeNewWindow('Image');
        Makeroi(0,0,newwidth,newheight);}
        {selectall;}
        paste;
        killroi;
        enhanceContrast;
        ApplyLUT;
        RestoreState;
        setcounter(slices-1);
        showresults;
        SelectWindow('results');
        TimeTaken:=round((TickCount-StartTime)/60);
        if not VerReliable then
                putmessage('Vertical Overlap unreliable. Vertical variance =
                        ',VerVariance);
        if not HorReliable then
                putmessage('Horizontal shift unreliable. Horizontal variance =
                        ',HorVariance);
        putmessage('Time Taken in seconds= ',TimeTaken);
end;


macro 'TestBlock[T]';
var
        x,y,range: real;
        left,top: integer;

begin
        x:=50;
        y:=50;
        range:=0;
        resetCounter;
        setOptions('User2');
        Usercode('test',range,x,y);
        left:=rUser1[1];
        Top:=rUser1[2];
        ChooseSlice(2);
        makeRoi(left,top,x,y);
        setUser2Label('variance');
        UpdateResult;
end;
```

APPENDIX C- PASCAL SOURCE CODE

```
unit User;
interfaceuses
        Memory, QuickDraw, Packages, Menus, Events, Fonts, Scrap, ToolUtils, Resources,
Errors,  Palettes, globals, Utilities, Graphics, Filters, Analysis;

procedure
        UserMacroCode (str: str255; Param1, Param2, Param3: extended);

implementation

Type
        Block=array[0..50] of array[0..50] of byte;

begin

{procedure to match two partially overlapping  images.}
procedure Match(strip, MaxHshift, MaxVshift: Extended);

var
        i1, i2, sum, difference, i1Inv: LineType;
        i, slice, row, row1, row11, j, k, l, Hshift, counter: integer;
        SumOfDiff, MinSumOfDiff: LongInt;
        mask: rect;
        overlapstr,Vendstr,Vstartstr,Hstartstr,Hendstr,Hshiftstr,Hrangestr:str255;
        overlap, MaxInt: LongInt;
        MaxVshiftInt, stripInt, MaxHshiftInt, HalfMaxHShift, temp,measurecount: integer;

        Hstart,Hend,Vstart,Vend,Hrange,i2index: integer;
        start:boolean;

begin
        start:=true;
        MaxInt := 65535;
        MaxHshiftInt := trunc(MaxHshift);
        stripInt := trunc(strip);
        MaxVshiftInt := trunc(MaxVshift);
        HalfMaxHShift := trunc(MaxHshiftInt / 2);
        Vstart:=trunc(User1^[3]);
        Vend:=trunc(User1^[4]);
        RealToString(Vend, 5, 2, Vendstr);
        RealToString(Vstart, 5, 2, Vstartstr);
        Hshift:=trunc(User1^[5]);
        Hrange:=trunc(User1^[6]);
        RealToString(Hshift, 5, 2, Hshiftstr);
        RealToString(Hrange, 5, 2, Hrangestr);
        i2index:=(Hrange div 2) - Hshift;
        with info^ do
        begin
                if StackInfo = nil then
                        exit(Append);
                counter := 0;
                for row1 := Vstart-stripInt to Vend-StripInt  do
```

```
                begin                                   {top level strip vertical movement}
                        ShowWatch;
                        for l := 0 to Hrange - 1 do
                        begin                           {adjust horizontal movement}
                                counter := counter + 1;
                                SumOfDiff := 0;
                                for row := nLines - stripInt to nLines - 1 do
                                begin                   {Process rows of strip}
                                        SelectSlice(1);
                                        row11 := row1 + row - nlines + stripInt;
                                        GetLine(0, row11, PixelsPerLine, i1);
                                        SelectSlice(2);
                                        GetLine(0, row, PixelsPerLine, i2);
                                        for j := 0 to PixelsPerLine - 1 - Hrange do
                                        begin           {Process Pixels in each line of
strip}
                                                temp := abs(integer(i1[j + l] -
i2[j+i2index]));
                                                SumOfDiff := SumOfDiff + temp;
                                        end;            {for j...}
                                end;                    {for row....}
                                if start then
                                begin
                                        MinSumOfDiff:=SumOfDiff;
                                        start:=false;
                                        if User1^[1]=1 then
                                                for measurecount:=1 to 51 do measure;

                                end;
                                if SumOfDiff <= MinSumOfDiff then
                                begin                   {Check sum of difference}
                                        MinSumOfDiff := SumOfDiff;
                                        overlap := row1 + stripInt;
                                        Hshift := l-i2index;
                                end;
                                if User1^[1] = 1 then
                                begin                   {measure;}
                                        ComputeResults;
                                        AppendResults;
                                        UpdateList;
                                        User1^[counter + 50] := row1+StripInt;
                                        User2^[counter + 50] := l-i2index;
                                        Mean^[counter + 50] := SumOfDiff;
                                end
                        end;
                        if CommandPeriod then
                                leave;
                end;
        end;            {with}
        User1^[1] := overlap;
        User1^[2] := Hshift;
{       RealToString(overlap, 5, 2, overlapstr);
        RealToString(Hshift, 5, 2, Hshiftstr);
        ShowMessage(concat('Vstart=', Vstartstr, cr, 'Vend=', Vendstr,cr,'Hrange=',
        Hrangestr,cr,'overlap=', overlapstr, cr, 'Hshift=', Hshiftstr));}
end;
```

```
function FindFeatureBlockRange(Range,m,n:integer; var LeftBlock,TopBlock:integer):Block;

var
        TempBlock:Block;
        width,Height,RightShift,DownShift,Vloc:integer;
        Line:LineType;
        pixel:integer;
        NoOfPixels:integer;
        LineSum,BlockSum:LongInt;
        LineSumSqr,BlockSumSqr:LongInt;
        variance,MaxVariance:LongInt;
        mstr,nstr,Downshiftstr,RightShiftstr,NoOfPixelsStr,widthstr,heightstr,tempstr:Str255;
        temp:integer;
        first:integer;

begin
        SelectSlice(2);
        with info^.PicRect do
        begin
                width:=right - left;
                height:= bottom - top;
        end;
        NoOfPixels:=m*n;
        MaxVariance:=0;
        first:=0;
        Downshift:=(Height-1-N-((Range-1)*N);
        While Downshift < Height-1-N Do
        begin
                if first <3  then
                temp:=DownShift;
                Numtostring(temp,tempstr);
                first:=first+1;
                Numtostring(Downshift,Downshiftstr);
                showmessage(concat('DownShift= ',Downshiftstr,cr,'temp=',tempstr));
                for RightShift:=0 to width - m -1 do
                Rightshift:=0;
                While Rightshift < Width-M-1 Do
                begin                                   {move block to the right}
                        {Numtostring(m, mstr);
                        NumToString(n, nstr);
                        ShowMessage(concat('m=',mstr,cr,'n=',nstr,cr,'in the feature
                                routine'));
                        Numtostring(width,widthstr);Numtostring(height,heightstr);
                        Numtostring(RightShift,RightShiftstr);
                        Numtostring(RightShift,RightShiftstr);
                        Numtostring(NoOfPixels,NoOfPixelsStr);
                        putmessage(concat('DownShift= ',Downshiftstr,cr,'RightShift=
                                        ',RightShiftstr,
                        cr,'NoOfPixels=',NoOfPixelsStr,cr,'width=',widthstr,cr,'height=',
                                heightstr));}
                        BlockSumSqr:=0;
                        BlockSum:=0;
                        for Vloc:=DownShift to DownShift+ n do
                        begin                           {process lines of block}
                                LineSumSqr:=0;
                                LineSum:=0;
                                getline(RightShift,Vloc,m-1,Line);
                                for pixel:=0 to m-1 do
```

```
                                begin                    {Process pixels within lines of
block}
                                        LineSumSqr:=LineSumSqr +
                                                        LongInt(sqr(Line[pixel]));
                                        LineSum:=LineSum + (Line[pixel]);
                                        TempBlock[Vloc-height+n ,Pixel]:=Line[Pixel];
                                end;
                                BlockSumSqr:=BlockSumSqr + LineSumSqr;
                                BlockSum:=BlockSum + LineSum;
                        end;

                        Variance:=(BlockSumSqr  div NoOfPixels) - sqr(BlockSum div
                                                NoOfPixels);
                        {ComputeResults;
                        AppendResults;
                        UpdateList;
                        User1^[RightShift] := Vloc;}
                        if Variance > MaxVariance then
                        begin
                                MaxVariance:= Variance;
                                LeftBlock:=RightShift;
                                TopBlock:=Downshift;
                                FindFeatureBlockRange:=TempBlock;
                        end;
                Rightshift:=Rightshift+((M-1)/2)
                end;
        Downshift=Downshift+((N-1)/2);
        end;
end;


function FindFeatureBlock(m,n:integer; var offset:integer):Block;
var
        TempBlock:Block;
        width,Height,RightShift,Vloc:integer;
        Line:LineType;
        pixel:integer;
        NoOfPixels:integer;
        LineSum,BlockSum:LongInt;
        LineSumSqr,BlockSumSqr:LongInt;
        variance,MaxVariance:LongInt;
        mstr,nstr:Str255;

begin
        SelectSlice(2);
        with info^.PicRect do
        begin
                width:=right - left;
                height:= bottom - top;
        end;

        NoOfPixels:=m*n;
        MaxVariance:=0;

        for RightShift:=0 to width - m -1 do
        begin                                   {move block to the right}
                BlockSumSqr:=0;
                BlockSum:=0;
```

```
                for Vloc:=height -n  to height -1 do
                begin                                   {process lines of block}
                        LineSumSqr:=0;
                        LineSum:=0;
                        getline(RightShift,Vloc,m-1,Line);
                        for pixel:=0 to m-1 do
                        begin                           {Process pixels within lines of block}
                                LineSumSqr:=LineSumSqr + LongInt(sqr(Line[pixel]));
                                LineSum:=LineSum + (Line[pixel]);
                                TempBlock[Vloc-height+n ,Pixel]:=Line[Pixel];
                        end;
                        BlockSumSqr:=BlockSumSqr + LineSumSqr;
                        BlockSum:=BlockSum + LineSum;
                end;
                Variance:=(BlockSumSqr  div NoOfPixels) - sqr(BlockSum div NoOfPixels);
                {ComputeResults;
                AppendResults;
                UpdateList;
                User1^[RightShift] := Vloc;}
                if Variance > MaxVariance then
                begin
                MaxVariance:= Variance;
                offset:=RightShift;
                FindFeatureBlock:=TempBlock;
                end;
        end;
        RealToString(m, 5, 2, mstr);
        RealToString(n, 5, 2, nstr);
        ShowMessage(concat('m=',mstr,cr,'n=',nstr,cr,'in the feature routine'));
end;


{x and y are the horizontal and vertical sizes of the block to be searched for and matched}
{range is the range to be searched from the bottom of image}
Procedure BlockMatch(range,x, y: Extended);
var
        m,n:integer;
        feature:Block;
        Hshift,counter: integer;
        SumOfDiff, MinSumOfDiff,temp: LongInt;
        overlapstr,Vendstr,Vstartstr,Hstartstr,Hendstr,Hshiftstr,Hrangestr:Str255;
        Offsetstr,mstr,nstr:str255;
        overlap:Integer;
        measurecount: integer;
        Hstart,Hend,Vstart,Vend,Hrange,i2index: integer;
        Hloc,Vloc:integer;
        FeatureRow,FeatureColumn:integer;
        column,row:integer;
        ImagePixel,FeaturePixel:integer;
        width,height:integer;
        FeatureColumnstr,FeatureRowstr,rowstr,columnstr,TopBlockstr:str255;
        dummy:integer;
        canceled:boolean;
        i2counter:integer;
        i1,i2:LineType;
        i,j:integer;
        leftBlock,TopBlock,Top:integer;
        irange:integer;
```

```
        Widthstr,Heightstr:str255;

begin
        with info^ do
        begin
                if StackInfo = nil then
                        exit(BlockMatch);
        end;
        MinSumOfDiff:=999999999;
        Vstart:=trunc(User1^[3]);
        Vend:=trunc(User1^[4]);
        Hshift:=trunc(User1^[5]);
        Hrange:=trunc(User1^[6]);
        irange:=trunc(range);
        m:=trunc(x);
        n:=trunc(y);
        Feature:=FindFeatureBlockRange(irange,m,n,leftBlock,topBlock);
        SelectSlice(1);
        with info^.PicRect do
        begin
                width:=right - left;
                height:= bottom - top;
        end;
        RealToString(Hshift, 5, 2, Hshiftstr);
        RealToString(LeftBlock, 5, 2, Offsetstr);
        Hstart:=leftblock - (Hrange div 2) + Hshift;
        if  Hstart<0 then
                Hstart:=0;
        Hend:=Hstart+Hrange;
        if  Hend>(width-1) -m then
                Hend:=(width-1) -m;
{       RealToString(Hend, 5, 2, Hendstr);
        RealToString(Width, 5, 2, Widthstr);
        RealToString(Height, 5, 2, Heightstr);
        putmessage(concat('hend=',Hendstr,cr,'Width=',widthstr,cr,'Height=',heightstr));}

        for Hloc:=Hstart to Hend do
        begin                                   {move block to the right}
                for Vloc:=Vstart to Vend do
                begin                           {move block downwards}
                        i2counter:=0;
                        SumOfDiff:=0;
                        for i:=Vloc to Vloc+n-1 do
                        begin
                                inc(i2counter);
                                SelectSlice(1);
                                GetLine(Hloc,i, m, i1);
                                SelectSlice(2);
                                Top:=(height-1)-n+i2counter-(height-1-n-topBlock);
                                GetLine(leftBlock,Top, m, i2);
                                for j := 0 to m - 1 do
                                begin           {Process Pixels in each line of strip}
                                        temp := abs(integer(i1[j] - i2[j]));
                                        SumOfDiff := SumOfDiff + temp;
                                end;    {for j...}
                        end;

                        if SumOfDiff <= MinSumOfDiff then
```

```
                        begin                                   {Check sum of difference}
                                MinSumOfDiff := SumOfDiff;
                                overlap := Vloc+n-1;
                                Hshift := Hloc-LeftBlock;
                        end;
                        if User1^[1] = 1 then
                         begin                                  {measure;}
                                for measurecount:=1 to 51 do measure;
                                ComputeResults;
                                AppendResults;
                                UpdateList;
                                User1^[counter + 50] := Vloc;
                                User2^[counter + 50] := Hloc;
                                Mean^[counter + 50] := SumOfDiff;
                        end;
                        if CommandPeriod then
                                leave;
                end;                            {Vloc}
        end;                                    {Hloc}
        User1^[1] := overlap;
        User1^[2] := Hshift;
        RealToString(Hstart, 5, 2, Hstartstr);
        RealToString(Hend, 5, 2, Hendstr);
        RealToString(overlap, 5, 2, overlapstr);
        RealToString(Vend, 5, 2, Vendstr);
        RealToString(Vstart, 5, 2, Vstartstr);
        RealToString(Hrange, 5, 2, Hrangestr);
        RealToString(m, 5, 2, mstr);
        RealToString(n, 5, 2, nstr);
        ShowMessage(concat('m=',mstr,cr,'n=',nstr,cr,'Vstart=', Vstartstr, cr, 'Vend=',
                Vendstr,cr,'offset= ',offsetstr,cr,'Hrange=', Hrangestr,cr,'overlap=', overlapstr,
                cr, 'Hshift=',  Hshiftstr,cr,'Hstart',Hstartstr,cr,'Hend',Hendstr));
end;


Procedure TestFeatureBlock(range, x, y: Extended);
var
        irange,m,n:integer;
        feature:Block;
        mstr,nstr,leftstr,topstr:Str255;
        BLeft,BTop:integer;

begin
        with info^ do
        begin
                if StackInfo = nil then
                        exit(TestFeatureBlock);
        end;
        irange:=trunc(range);
        m:=trunc(x);
        n:=trunc(y);
        Feature:=FindFeatureBlockRange(irange,m,n,BLeft,BTop);
        NumToString(m, mstr);
        NumToString(n, nstr);
        NumToString(Bleft, leftstr);
```

```
        NumToString(Btop, topstr);
        ShowMessage(concat('m=',mstr,cr,'n=',nstr,cr,'left=',leftstr,cr,'top=',topstr,cr,'in the
                Test routine'));
        User1^[1] := BLeft;
        User1^[2]:=BTop;
end;


procedure UserMacroCode (str: str255; Param1, Param2, Param3: extended);
begin
        MakeLowerCase(str);
        if pos('Match', str) <> 0 then
        begin
                Match(Param1, Param2, Param3);
                exit(UserMacroCode);
        end;

        if pos('BlockMatch', str) <> 0 then
        begin
                BlockMatch(Param1, Param2,Param3);
                exit(UserMacroCode);
        end;

        if pos('test', str) <> 0 then
        begin
                TestFeatureBlock(Param1, Param2,Param3);
                exit(UserMacroCode);
        end;
        ShowNoCodeMessage;
end;

end.
```

APPENDIX D- FFT AND FHT

FOLLOWING IS AN EXCERPT FROM A REPORT BY

A. ARLO REEVES

THAYER SCHOOL OF ENGINEERING

DARTMOUTH COLLEGE

OPTIMIZED FAST HARTLEY TRANSFORM FOR THE MC68000 WITH APPLICATIONS IN IMAGE PROCESSING

2. MATHEMATICAL FOUNDATIONS

Having narrowed the scope of interest down to the Hartley transform, more mathematical details can now be given. To provide a standard against which to compare the Hartley transform, we introduce it in parallel with the Fourier transform.

2.1 The Fourier and Hartley Transforms

A physical process can be described either in the *time domain* by some function V(t) or in the *frequency domain* by some (generally complex) function F(f). Both descriptions contain the same information and can therefore be thought of as different representations of the same function [14]. To produce one representation of the function from the other, one uses the *Fourier transform* and the *inverse Fourier transform*[1]:

$$F(f) = \int_{-\infty}^{\infty} V(t)\, e^{-2\pi i f t}\, dt$$

$$V(t) = \int_{-\infty}^{\infty} F(f)\, e^{2\pi i f t}\, df \qquad (1)$$

The Hartley transform and its inverse are very similar to their Fourier counterparts [15]:

$$H(f) = \int_{-\infty}^{\infty} V(t)\, \mathrm{cas}(2\pi f t)\, dt$$

$$V(t) = \int_{-\infty}^{\infty} H(f)\, \mathrm{cas}(2\pi f t)\, df \qquad (2)$$

where the cas function is defined

$$\mathrm{cas}(t) \equiv \cos(t) + \sin(t). \qquad (3)$$

Comparing the Fourier and Hartley transform pairs, two distinctions are immediately evident. First, the Hartley transform of a real valued function is itself real valued.

[1]The Fourier transform of V(t) exists if V(t) it is bounded and absolutely integrable [4].

Second, there is absolutely no difference between the forward and inverse Hartley transform. These two characteristics, combined with the fact that the Fourier transform of a real function can be easily derived from its Hartley transform, make the Hartley transform an efficient and convenient vehicle for Fourier transform calculation, as will be shown below.

The relationship between the Fourier and Hartley transforms hinges upon symmetry conditions. Splitting the Hartley transform $H(f)$ into its even and odd components, $E(f)$ and $O(f)$, we obtain [15]

$$E(f) = \frac{H(f) + H(-f)}{2} = \int_{-\infty}^{\infty} V(t) \cos(2\pi f t)\, dt$$

$$O(f) = \frac{H(f) - H(-f)}{2} = \int_{-\infty}^{\infty} V(t) \sin(2\pi f t)\, dt \qquad (4)$$

From these relations, we see that the Fourier transform can be obtained from the Hartley transform by forming the difference $E(f) - iO(f)$:

$$E(f) - iO(f) = \int_{-\infty}^{\infty} V(t)(\cos 2\pi f t - i \sin 2\pi f t)\, dt$$

$$= \int_{-\infty}^{\infty} V(t)\, e^{-i2\pi f t}\, dt$$

$$= F(f) \qquad (5)$$

Conversely, the Hartley transform can be obtained from the Fourier transform by computing

$$H(f) = F_{real}(f) - F_{imaginary}(f). \qquad (6)$$

The real and imaginary parts of a signal's Fourier transform are often of less interest than their derivates: The signal's amplitude, phase and power spectrum. Using the definitions of these quantities and the equations (4) and (5), these values can be computed directly from both the Fourier and Hartley transforms as follows:

Function	Fourier Calculation	Hartley Calculation
Power Spectrum	$[F_r(f)]^2 + [F_i(f)]^2$	$\frac{[H(f)]^2 + [H(-f)]^2}{2}$
Amplitude	$\sqrt{[F_r(f)]^2 + [F_i(f)]^2}$	$\sqrt{[H(f)]^2 + [H(-f)]^2}$
Phase	$\arctan\left[\frac{F_i(f)}{F_r(f)}\right]$	$\arctan\left[\frac{H(-f)}{H(f)}\right] + \frac{\pi}{8}$

Table 1: Power Spectrum, Magnitude and Phase as calculated from the Fourier and Hartley transforms [15].

As Table 1 clearly shows, the calculations involved in recovering the power spectrum, amplitude and phase of a signal from its Fourier or Hartley transform are very similar and demand the same number of operations.

The similarity between the two transforms extends to the theorems commonly used in Fourier analysis. For virtually all Fourier transform theorems, there is a corresponding Hartley transform theorem. Both transforms are linear and therefore have Addition theorems (see Table 2). The Similarity theorem, which states that contraction of a signal's space domain representation corresponds to a dilation of its it frequency domain representation, also exists for the Hartley transform. This trend holds for the important Convolution and Correlation theorems as well, enabling these operations to be directly computed from the Hartley transform.

Theorem	V(t)	F(f)	H(f)
Addition	$V_1(t) + V_2(t)$	$F_1(f) + F_2(f)$	$H_1(f) + H_2(f)$
Similarity	$V(t/T)$	$\|T\|F(Tf)$	$\|T\|H(Tf)$
Shift	$V(t - T)$	$e^{-2\pi iTf/N}F(f)$	$\sin(2\pi Tf)H(-f) + \cos(2\pi Tf)H(f)$
Reversal	$V(-t)$	$F(-f)$	$H(-f)$
Convolution	$V_1(t) * V_2(t)$	$F_1(f)F_2(f)$	$\frac{1}{2}[H_1(f)H_2(f) - H_1(-f)H_2(-f) + H_1(f)H_2(-f) + H_1(-f)H_2(f)]$
Correlation	$V_1(t)$ H $V_2(t)$	$F_1(f)[F_2(f)]^*$	$\frac{1}{2}[H_1(f)H_2(f) + H_1(-f)H_2(-f) + H_1(f)H_2(-f) - H_1(-f)H_2(f)]$

H Denotes Correlation

Table 2: Theorems for the Fourier and Hartley transforms [15].

The strong similarity between the Hartley and Fourier transforms enables them to be interchanged in most situations. As we will see, making this exchange is worthwhile when computing the Fourier transform of real sequences.

2.2 The Discrete Fourier & Hartley Transform

While people tend to think in terms of continuous variables, it is usually necessary to use discrete variables when making measurements and performing computations. Computing the Fourier transform is no exception, so a discrete form of the Fourier transform and its inverse is needed:

$$F(k) = \sum_{n=0}^{N-1} V(n)\, e^{-2\pi ikn/N}$$

$$V(n) = \frac{1}{N}\sum_{n=0}^{N-1} F(k)\, e^{2\pi ikn/N} \qquad (7)$$

Here, our continuous function of time, V(t), has been sampled at N equispaced points producing a discrete sequence of N numbers, V(n). The Fourier transform, F(k), has

likewise become a discrete sequence consisting of N complex numbers[2]. The discrete Fourier transform (DFT) is very similar to is continuous counterpart, except for the factor of 1/N in the inverse transform. This factor is as often associated with the the forward transform as its inverse; while its location is not critical[3], it is necessary to restore the proper scale to a sequence that has been transformed and then inverse transformed [15]. Using the same discrete sequence, V(n), the discrete Hartley transform (DHT) and its inverse become:

$$H(k) = \sum_{n=0}^{N-1} V(n)\ \mathrm{cas}(2\pi kn/N)$$

$$V(n) = \frac{1}{N}\sum_{n=0}^{N-1} H(k)\ \mathrm{cas}(2\pi kn/N) \tag{8}$$

Except for a factor of 1/N, the DHT and its inverse are again identical. The relation between the DFT and DHT is also analogous to the continuous case; the DFT may be obtained from the DHT using

$$F(k) = E(k) - iO(k) \tag{9}$$

where

$$E(k) = \frac{H(k) + H(N-k)}{2}$$

$$O(k) = \frac{H(k) - H(N-k)}{2} \tag{10}$$

Conversely, the DHT can be computed from the DFT using

$$H(k) = F_r(k) - F_i(k) \tag{11}$$

as before. The direct analogy between the discrete and continuous cases also extends to the computations of amplitude, phase and power spectra, the only difference being that $H(-f)$ is replaced with H(N-k)[4]. Likewise, the discrete theorems are very similar, except

---

[2]In accordance with popular convention, we denote these indices as ranging from 0 to N-1, with the result that positive frequencies correspond to k in the range [1..N/2-1], while negative frequencies correspond to the 'second half of the array' or k in [N/2+1..N-1]. Note also that k = 0 is the 'DC' or zero frequency component of the spectrum, while k = N/2 is at once the maximum positive and negative frequency in the spectrum (these are equal due to the spectrum's periodicity).

[3]The transform pair can even be made symmetrical by multiplying both sums by $1/\sqrt{N}$.

[4]Which arises from letting k range from 0 to N-1, and not from -N/2 to N/2.

in two important cases which underscore the difference between the continuous and discrete transforms: The Convolution and Correlation theorems.

Discrete sampling of a signal can be modeled as multiplying the signal by a sequence of delta functions spaced some sampling interval, ΔT, apart. The transform of this delta function sequence is itself a delta function sequence, spaced $1/\Delta T$ apart in the frequency domain. Since multiplication in the time domain is equivalent convolution in the frequency domain, the sampling of the signal results in the periodic reproduction of its transform with a $1/\Delta T$ spacing in the frequency domain[5]. Likewise, the sampling or discretization of the transform results in the signal being periodically extended in the space domain. Consequently the signal, its DFT and its DHT are all periodic sequences with period N, as equations (7) and (8) will verify.

The discrete convolution and correlation theorems therefore describe cyclic convolution and cyclic correlation: In contrast to the continuous case, the shift and multiply operation characterizing convolution and correlation produces overlap between the sampled signal and its periodic extension. Consequently, the cyclic convolution and cyclic correlation are also periodic functions[6].

[5]To develop a graphical understanding of this, read E. O. Brigham's 'Graphical Development of the Discrete Fourier Transform' in his fine book, *The Fast Fourier Transform* [4].

[6]If one wishes to compute a normal, non-cyclic convolution or correlation, this overlap effect can be avoided by padding the data with zeros, as is described in Appendix A

Theorem	V(n)	F(k)	H(k)
Addition	$V_1(n) + V_2(n)$	$F_1(k) + F_2(k)$	$H_1(k) + H_2(k)$
Shift	$V(n - T)$	$e^{-2\pi iTk/N}F(k)$	$\cos(2\pi Tk/N)H(k) - \sin(2\pi Tk/N)H(N-k)$
Reversal	$V(-n)$	$F(N-k)$	$H(N-k)$
Convolution	$V_1(n)$ ¿ $V_2(n)$	$NF_1(k)F_2(k)$	$\frac{1}{2}N[H_1(k)H_2(k) - H_1(N-k)H_2(N-k) + H_1(k)H_2(N-k) + H_1(N-k)H_2(k)]$
Correlation	$V_1(n)$¡ $V_2(n)$	$NF_1(k)[F_2(k)]^*$	$\frac{1}{2}N[H_1(k)H_2(k) + H_1(-k)H_2(N-k) + H_1(k)H_2(N-k) - H_1(N-k)H_2(k)]$

¿ Denotes cyclic convolution.

¡ Denotes cyclic correlation.

Table 3: Theorems for the DFT and DHT[7] [15].

Sampling also limits the accuracy with which a DFT models the continuous Fourier transform sought. Because of the transform's duplication every $1/\Delta T$ cycles, it is necessary that the transform go to zero for frequencies exceeding $1/2\Delta T$ (the Nyquist Frequency). Frequencies higher than this are *aliased* into the interval $\pm 1/2\Delta T$. Furthermore, since sequences always have finite length, their transforms are convolved with with the transform of their windowing function, decreasing the spectral resolution in an effect known as *leakage* [4]. Once aware of these limitations of the DFT and DHT, however, one can minimize their effects.

The Two Dimensional DFT and Two Dimensional DHT

Until now we have considered only one dimensional signals and sequences in the time domain. Images are two dimensional *space domain* objects. The only difference between the two domains is that the temporal causality of the time domain is absent in the space domain; every pixel of an image can be accessed simultaneously, whereas a time series is inherently sequential.

[7]For the discrete Similarity theorem, see Bracewell [15].

While white light's rainbow colored spectrum is an easily visualized manifestation of Fourier transformation in one dimension, how does one visualize a two dimensional Fourier transform? As a diffraction pattern; if an object is considered to be a two dimensional aperture function, A(y,z), its far-field Fraunhofer diffraction pattern is the Fourier transform of A(y,z) [19]. If this description does little for your intuition, then using the FFT extensions to *Image* will; there is no better way to develop this intuition than by direct experience.

The two dimensional DFT and its inverse are expressed:

$$F(k_1,k_2) = \sum_{n_2=0}^{N_2-1}\sum_{n_1=0}^{N_1-1} V(n_1,n_2)\exp\left[-2\pi i\left(\frac{k_1n_1}{N_1}+\frac{k_2n_2}{N_2}\right)\right]$$

$$V(n_1,n_2) = \frac{1}{N_1N_2}\sum_{n_2=0}^{N_2-1}\sum_{n_1=0}^{N_1-1} F(k_1,k_2)\exp\left[2\pi i\left(\frac{k_1n_1}{N_1}+\frac{k_2n_2}{N_2}\right)\right] \qquad (12)$$

Because of the exp function's separability ($e^Ae^B = e^{(A+B)}$), the two dimensional (2D) DFT can be computed as the DFT of one dimensional DFTs:

$$F(k_1,k_2) = \sum_{n_2=0}^{N_2-1}\left[\sum_{n_1=0}^{N_1-1} V(n_1,n_2)\exp\left[-2\pi i\left(\frac{k_1n_1}{N_1}\right)\right]\right]\exp\left[-2\pi i\left(\frac{k_2n_2}{N_2}\right)\right]$$

$$V(n_1,n_2) = \frac{1}{N_2}\sum_{n_2=0}^{N_2-1}\left[\frac{1}{N_1}\sum_{n_1=0}^{N_1-1} F(k_1,k_2)\exp\left[2\pi i\left(\frac{k_1n_1}{N_1}\right)\right]\right]\exp\left[2\pi i\left(\frac{k_2n_2}{N_2}\right)\right] \qquad (13)$$

This means that the 2D DFT of an image, for example, can be computed by first transforming each of the rows of the image and then transforming each of the columns of the image or vice versa.

The 2D DHT is expressed

$$F(k_1,k_2) = \sum_{n_2=0}^{N_2-1}\sum_{n_1=0}^{N_1-1} V(n_1,n_2)\,\mathrm{cas}\left[2\pi\left(\frac{k_1n_1}{N_1}+\frac{k_2n_2}{N_2}\right)\right]$$

$$V(n_1,n_2) = \frac{1}{N_1N_2}\sum_{n_2=0}^{N_2-1}\sum_{n_1=0}^{N_1-1} F(k_1,k_2)\,\mathrm{cas}\left[2\pi\left(\frac{k_1n_1}{N_1}+\frac{k_2n_2}{N_2}\right)\right] \qquad (14)$$

As in the one dimensional case, the forward and inverse transforms are identical but for a multiplicative factor. Unfortunately, however, the cas function is not separable and

therefore the 2D DHT cannot be computed by simply applying a 1D DHT to the rows and columns of the image matrix. If a row-column DHT is computed, however, the two dimensional DHT can be recovered from the result, as is described under 'Row Column HT to Two Dimensional HT Conversion' below.

2.3 The Fast Fourier Transform

Because computing the DFT of an N point sequence requires N summations each involving N operations, the total computation requires $O(N^2)$ operations. Writing out the entire computation by hand will show, however, that many of these operations are redundant and can be eliminated. Using Danielson and Lanczos' [3] observation that an N point DFT can be expressed as the summation of two N/2 point DFTs, these redundancies can be eliminated as we now show. Adopting the conventional definition

$$W \equiv e^{-2\pi i/N}, \tag{15}$$

where W is commonly referred to as a *twiddle factor*[8], the DFT can be divided in two as follows:

$$\begin{aligned} F(k) &= \sum_{n=0}^{N-1} V(n)W^{nk} \\ &= \sum_{n=0}^{N/2-1} V(2n)W^{(2n)k} + \sum_{n=0}^{N/2-1} V(2n+1)W^{(2n+1)k} \\ &= \sum_{n=0}^{N/2-1} V(2n)W^{(2n)k} + W^k \sum_{n=0}^{N/2-1} V(2n+1)W^{(2n)k} \\ &= F_{even}(k) + W^k F_{odd}(k) \end{aligned} \tag{16}$$

$F_{even}(k)$ is the N/2 point DFT of the even elements of V(n), while $F_{odd}(k)$ is the N/2 point DFT of the odd elements of V(n). The second N/2 points of the transform are likewise computed using

$$\begin{aligned} F(k + N/2) &= F_{even}(k + N/2) + W^{k+N/2}F_{odd}(k + N/2) \\ &= F_{even}(k) - W^k F_{odd}(k) \end{aligned} \tag{17}$$

since F_{even} and F_{odd} both have period N/2 and $W^{N/2} = -1$.

[8]The W^k are the N complex roots of unity and therefore lie on the unit circle in the complex plane.

While using this division reduces the total computation involved by almost a factor of 2, there is no reason to stop here. The beauty of the Danielson Lanczos Lemma is that it can be recursively applied to sub-sequences of length N/4, N/8, etc. [14]. When this technique is used on sequences an integer power of two in length, the division process can proceed $\log_2 N$ times, producing $\log_2 N$ *stages* each of which require O(N) operations. The resulting total operation count of $O(N\log_2 N)$ provides a vast improvement over $O(N^2)$ for large N, making this the *Fast Fourier Transform* or FFT.

The fundamental computational unit of the FFT is a two point transform, called a *butterfly* because of its appearance:

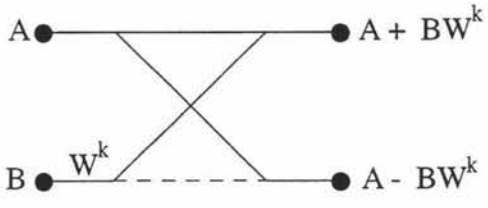


Figure 1: FFT Butterfly. The dashed line indicates negation.

Two input data points, A & B, produce two output points after one complex twiddle factor multiplication and two complex additions or a total of 4 real multiplications and 5 real additions. Since only two elements are accessed per butterfly, the same storage can be used for input and output, and the computation is performed *in place*. In the first stage of the FFT computation, N/2 such butterflies are applied to the input sequence producing N/2 two point transforms. The second and subsequent stages combine butterflies into groups of 2, 4, 8, etc. to perform longer transforms. This is depicted here in a butterfly flow diagram for a 16 point FFT:

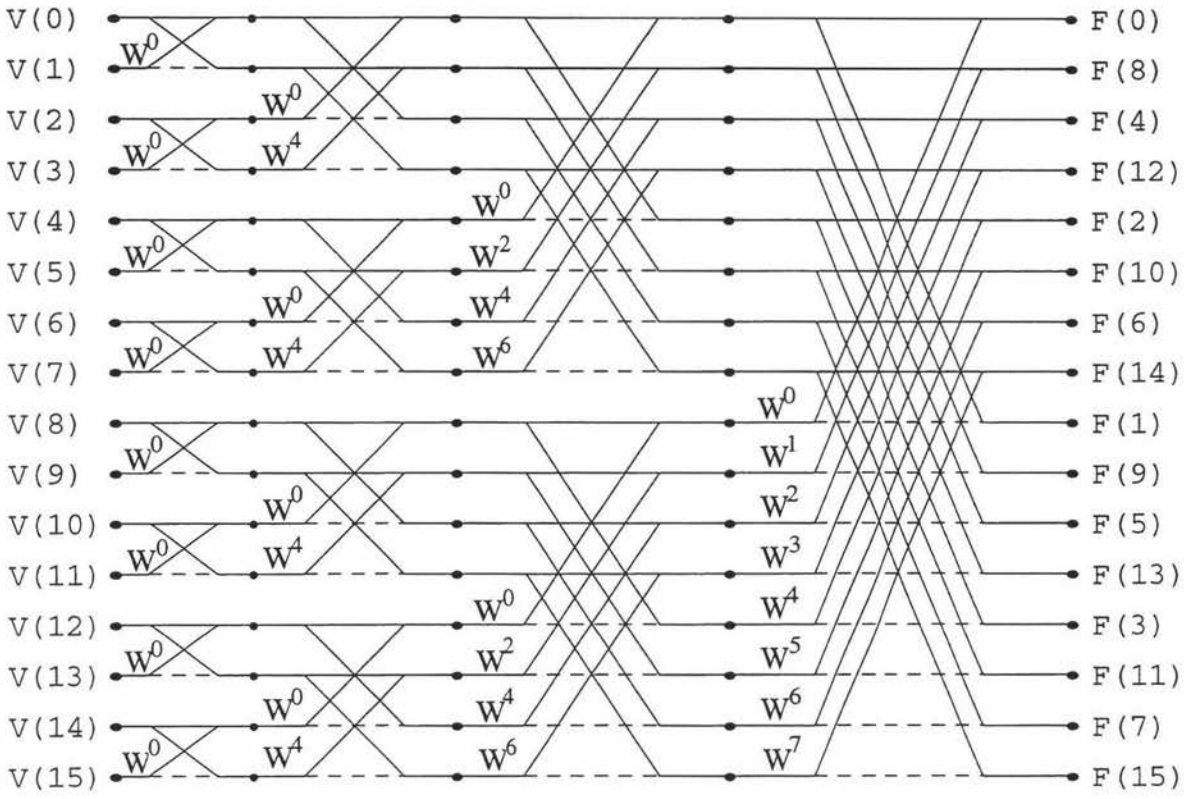


Figure 2: Butterfly flow diagram for a 16 point FFT.

The high degree of regularity in the butterfly diagram makes its implementation in code very compact. An FFT routine consists of three small loops, the outermost loop determines the stage and repeats $\log_2 N$ times, while the inner two loops control the number of butterflies in a group and the number of groups[9], together performing N/2 two point butterflies.

By repeatedly decimating the input into even and odd sub-sequences, the radix 2 FFT returns a permutation of the desired transform. To restore sequential ordering, the elements whose indices bit-wise mirror one another are swapped (e.g. the element at index 0011 (3) is swapped with the element at index 1100 (12))[10]. This *bit reversal* operation, described under 'Details of the Utility Routines' below, may be performed before or after the transform.

[9]This terminology of *stages*, *group size* and *number of groups* is carried through to the source code level.
[10]For higher radix algorithms, the indices are digit reversed in the radix used; for radix 4 permutations, the array elements whose base-four indices mirror one another are swapped. The general radix permutation is therefore described as digit-reversal [20].

The basic, radix-2 FFT algorithm is very symmetrical, but it accepts general complex input when all that we need here is the ability to transform real sequences. The Fourier transform of a real sequence has conjugate symmetry (the real part of the transform is even while the imaginary part is odd) which can be exploited to reduce the number of computations in an FFT by one half [14].

2.4 The Fast Hartley Transform

Like the DFT, the DHT summation can be split in two to reduce total operation count. Expressing equation (9) as

$$F(k) = \frac{H(k) + iH(N-k)}{1+i} \tag{18}$$

and substituting this expression into equation (16), we obtain

$$H(k) + iH(N-k) = [H_e(k) + iH_e(\frac{N}{2} -k)] + W^k[H_o(k) + iH_o(\frac{N}{2} -k)] \tag{19}$$

where $H_e(k)$ is the N/2 point DHT of the even indexed elements of H(k) and $H_o(k)$ is the N/2 point DHT of the odd indexed elements of H(k)[11]. Equating real and imaginary parts gives us the Hartley analog of equations (16) and (17) [21]:

$$\begin{aligned} H(k) &= H_e(k) + [H_o(k)\cos(2\pi k/N) + H_o((N/2) - k)\sin(2\pi k/N)] \\ H(k + N/2) &= H_e(k) - [H_o(k)\cos(2\pi k/N) + H_o((N/2) - k)\sin(2\pi k/N)] \end{aligned} \tag{20}$$

While k ranges from 0 to N-1, the indices for the even and odd sub-transforms are evaluated modulo N/2 [18]. Like the FFT, this decomposition can be recursively applied until length two transforms are obtained. Because the even-odd decimation is identical to that of the FFT, the bit reversal permutation is also the same.

Retrograde Indexing

In place computation is desired, yet equation (20) shows that both the kth element and the (N/2-k)th element of an N/2 point DHT must be accessed to compute one output point. Consequently, four elements must be processed at once to avoid overwriting an element that will be needed later. This is a manifestation of the FHT's *retrograde indexing* because the index, N/2-k, of the element multiplied by the sine term decreases with

[11]Not to be confused with E(k) and O(k), the even and odd parts of H(k).

increasing k while the indices for the other elements increase with k. This is most easily visualized in the structured butterfly flow diagram for the FHT [22]:

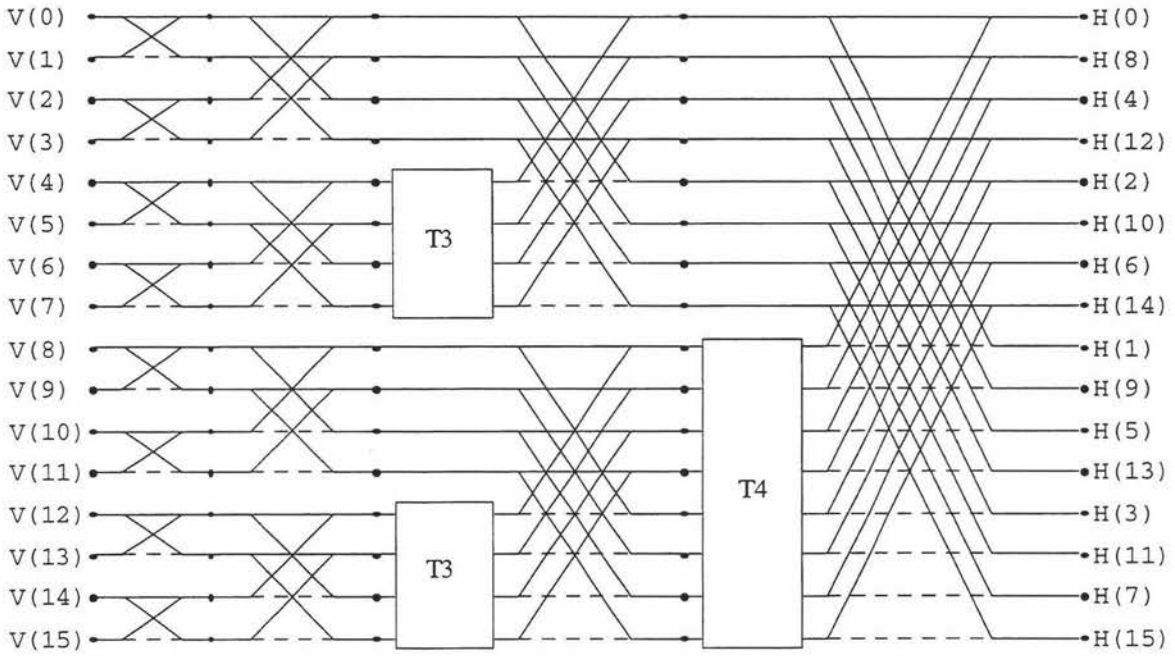


Figure 3: Structured Butterfly Flow Diagram for a 16 point FHT.

In Figure 3, butterflies operate in the same way as they do for the FFT. The absence of twiddle factors in the first stage is no mistake; the sine and cosine terms are either 1, 0, or -1 in these stages and are never both simultaneously non-zero.

The retrograde indexing becomes apparent in the third and subsequent stages, where the twiddle factor multiplication takes place in the boxes labeled T3 and T4. The data paths and twiddle factor multiplications inside these boxes are shown here:

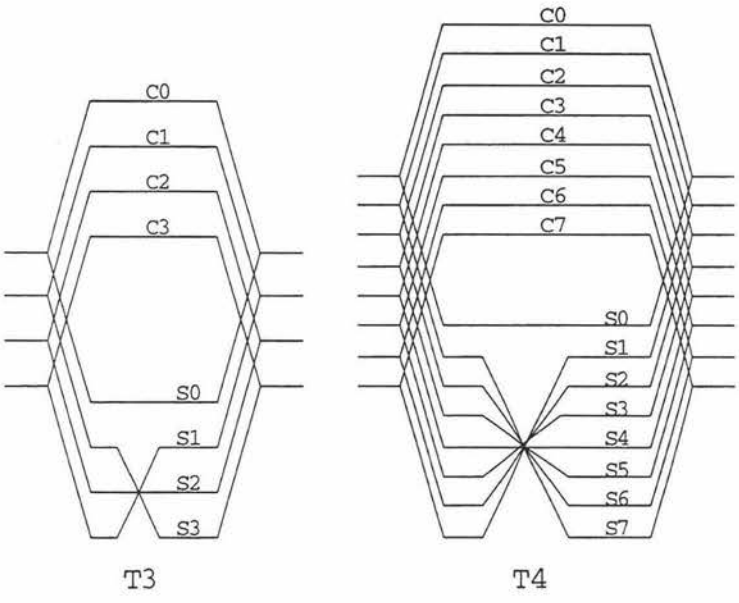


Figure 4: FHT Retrograde Indexed Twiddle Factor Multiplication.

In Figure 4, units T3 and T4 show how the twiddle factor multiplication takes place. The Ck and Sk denote the terms $\cos(2\pi k/N)$ and $\sin(2\pi k/N)$ respectively. The retrograde indexing of the sine terms can easily be seen to produce the requirement of two input points for each output point. To attain in place computation, two butterflies must therefore performed at once. By exploiting the symmetries

$$\sin\left(\frac{2\pi}{N}k(N - n)\right) = -\sin\left(\frac{2\pi}{N}kn\right)$$
$$\cos\left(\frac{2\pi}{N}k(N - n)\right) = \cos\left(\frac{2\pi}{N}kn\right), \quad (21)$$

the total number of multiplications for the 4 point dual-butterfly is reduced to a total of four real multiplications and six real additions [18]. Since two complex FFT butterflies require 8 real multiplications and 10 real additions, the operation count is reduced by almost one half.

Operation count of the FHT can be reduced yet further. Since the first two stages involve no multiplications, they can be computed separately. To minimize the number of memory accesses in the first two stages, they can be collapsed into one stage made up of N/4 radix 4 butterflies. The first butterfly of each group starting with the third stage likewise involves no multiplications and can be treated separately.

Because of the similarity between the four twiddle factor multiplications necessary to compute a dual-butterfly and a general complex multiplication, a trick to reduce the operation count in the latter computation can also be exploited. The product

$$(a + ib)(c + id) = (ac - bd) + i(ad + bc) = Re + iIm \tag{22}$$

which requires 4 multiplies and 3 adds can be computed with 3 multiplies and 5 adds by using a temporary variable T[12]:

$$\begin{aligned} T &= (a - b)d \\ Re &= T + a(c - d) \\ Im &= T + b(c + d) \end{aligned} \tag{23}$$

Further reductions in operation count can be achieved by exploiting the symmetries of the twiddle factors[13], but since they occur only under specific conditions, continuously checking for these conditions may outweigh the benefits of their 'optimization.'

[12]Because an ADD instruction runs between 7 and 18 times faster than a MUL instruction on the 680x0 processors, this is a worthwhile optimization.

[13]For example, when the sine and cosine terms are equal, a multiplication can be avoided.

APPENDIX E- SAMPLE STACK AND THE RESULTANT HIGH RESOLUTION IMAGE