AN ABSTRACT OF THE THESIS OF

FATIMA ASLAM for the degree of Master of Science in Electrical and Computer Engineering / Forest Products presented on June 6 , 1990.

Title: On-line Lumber Quality Control Technique Using Image Processing.

Abstract approved: Signature redacted for privacy.

Dr. James W. Funck

Product quality control is the back bone for an efficient process control system. By monitoring the size of the board cut in a sawmill, performance of the production system can be controlled. Feedback from the quality control system is used to evaluate the status of the production system and also used for preventive maintenance of the system. The software technique developed in this thesis involves three parts: image acquisition and analysis, quality control analysis, and power consumption analysis.

An image of the boards is acquired, and board edges are identified and analyzed to compute the board widths. The effects of defects present on the board surface on the computation of board width are studied. The effect of camera distance from the surface of the object is also analyzed. Computed widths were compared with values measured with a caliper.

The quality control program reads the computed board widths and then checks the measurements against control limits. The program also computes board averages and standard deviations and checks the averages against control limits. Between board deviations within cants and between saw pockets are also computed. The n board averages are displayed to check the process status.

A power meter is used to monitor the power consumption of the production system. A program was developed to read the power meter and to display the data in a control chart or in tabular form as a means of checking the power consumption of the system.

On-Line Lumber Quality Control Technique Using Image Processing

by

Fatima Aslam

A THESIS

Submitted to

Oregon State University

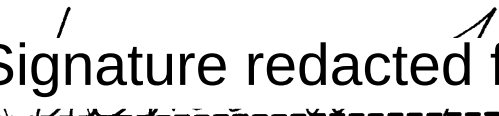
in partial fulfillment of
the requirements for the
degree of

Master of Science

Completed June 6, 1990

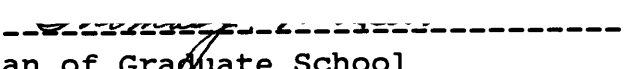
Commencement June 1991

APPROVED:



Associate Professor of Forest Products in charge of major

Head of Department of Forest Products



Dean of Graduate School

Date thesis presented: June 6, 1990

ACKNOWLEDGEMENTS

I would like to thank my major professor, Dr. James W. Funck, for his guidance, patience and understanding. It is by his great help I could sucessfully complete this thesis.

I would also like to thank Mr. Jeff Gaber and Mr. Milo Clauson for their help.

TABLE OF CONTENTS

INTRODUCTION 1

GOAL 3

OBJECTIVES 4

LITERATURE SURVEY 5

- Vision systems: 5
- Detecting an edge of an object: . 7
- Measuring an invariant of an object: . . 12
- Measurement error: 17
- Power consumption in the sawmill industry: 18
- Control chart: 19

PROJECT OVERVIEW 23

- Line-scan camera: 25
- Digitizing system: 25
- Power meter: 26
- Data acquisition system: 27

PROCEDURE 28

- Reading camera data: 30
- Plotting data: 33
- Computing board widths: 37
- Quality control program: 43
- Power measurement: 51

RESULTS AND DISCUSSION 55

Logging camera signal: 55
Plotting camera data points: . . . 58
Computing board widths: . . . 61
Quality control program: . . . 77
Power measurement: 87

CONCLUSIONS 96

BIBLIOGRAPHY 98

APPENDICES

A List of variables 100a
B Program listings 103a

LIST OF FIGURES

Figure		Page
1.	Gray level signal for four boards	8
2.	Measurement of board width	14
3.	Measurement configuration	15
4.	Tilted camera configuration	15
5.	A sample control chart	22
6.	Block diagram for the system used in this research.	24
7.	Plot of a six board cant	59
8.	Plot of a six board cant with knots near edges.	66
9.	Plot of a six board cant with a broken dark knot in the first board	68
10.	Plot of a six board cant. The edges between the first two boards are blocked with a piece of wood.	71
11.	Plot of a reference cant with 12 boards . .	74
12.	Plot of power consumption data	94

LIST OF TABLES

Table		Page
1.	Example of logged camera data for clear boards	56
2.	Program results for data in Table 1 . . .	60
3.	Computed and measured values for clear boards	62
4.	Results for boards with knots	67
5.	Results for a board with a dark broken knot .	69
6.	Results for a light colored broken knot .	72
7.	Results for the change of height	75
8.	Data file for the quality control program	78
9.	Quality control program initial questions list	79
10.	Results for all quality control check readings	81
11.	Computed board averages	83
12.	List of options menu	84
13.	Board standard deviations	85
14.	Results of computed control limits check	86
15.	Within cant between boards standard deviations . . .	88
16.	Between cant within saw pocket standard deviations	89

LIST OF FLOW DIAGRAMS

Flow diagram		Page
1.	Overview of quality control system	29
2.	Reading the camera data	32
3.	Plotting the camera data	34
4.	Computing board widths	38
5.	Computing board averages and standard deviations and checking for control limits . . .	45
6.	Computing control limits and checking averages . . .	46
7.	Computing within cant between board standard deviations	48
8.	Computing within saw pocket board standard deviations	49
9.	Computing n board averages	50
10.	Power measurement flowchart	53

ON-LINE LUMBER QUALITY CONTROL TECHNIQUE USING IMAGE PROCESSING

INTRODUCTION

An automated noncontact measurement system helps an industry to inspect the product and control the process on a real time basis without personal risk. In sawmills, checking the process accuracy has meant manually removing lumber from the system for measurement. If an automated system is developed to measure the characteristics of the product for inspection, it can be an aid for preventive maintenance as well as control of the process. The aim of this research project was to develop a technique which could be used to control the lumber production process in a sawmill. The waste produced during production could be reduced and lumber production improved.

A computer software system was developed to measure board width on a real time basis using a line-scan camera system. A quality control program for the measured width of the board was developed which reports

back the status of the process to monitoring personnel on a real time basis. It will be a very useful tool to aid decisions regarding preventive maintenance and control of the process.

The camera system scans the boards line-by-line and sends the analog signal to an oscilloscope where the signal is digitized. The digitized data is transferred to the computer where a software system manipulates the data to calculate the width of the boards. The measured board widths are used to calculate different statistics which are checked for their acceptance level by the quality control system and the status can be reported in different formats by selection.

Power consumption of the saw is continuously monitored either by a plot or by data displayed on the screen. Over time, the saw teeth dull resulting in an increase in power consumption. When this value goes up and above the optimum value fixed by the particular process environment, it will be reported so that the required maintenance action can be taken.

GOAL

Develop the software and interface the hardware required to provide real time quality control information after a multi-saw operation in a sawmill.

OBJECTIVES

The objectives are to:

- Interface a microcomputer with the hardware necessary to acquire the signal from a camera system and data from a power meter.

- Develop a software technique to compute the width of a board from the image captured by the camera system and develop a quality control technique for the computed measurement and power meter data.

LITERATURE SURVEY

The literature related to this subject covers a wide range of fields, from digital image processing to statistical quality control.

VISION SYSTEMS:

Industrial 2-D vision systems use solid state cameras with charge-coupled-device (CCD) linear arrays on their image planes. The linear arrays consist of photosensitive elements, called pixels, each with an area of about 13 x 13 micrometers. A pixel is a photocapacitor in which an electrical energy equivalent to the light energy falling on its surface is stored. The stored energy is a function of the light intensity, duration of the light level, and the pixel sensitivity. The voltage on a given pixel ranges between 0 and 2 volts. The analog signal from a pixel is transferred to a parallel to serial shift register and from there to a signal processor. Since the signal is transferred serially from the camera to the processor, the system is

slowed down irrespective of the camera's scanning speed. This can result in loss of data in real time measurement.

The geometry of an object can be derived from the 2-D image obtained by the camera. The gray level signals corresponding to each element of the linear array are converted to digital signals using an analog to digital converter. The digital signal represents the number of states or gray levels depending upon the number of bytes used to digitize the analog signal. For instance, an 8-bit system would be able to represent 256 distinct levels.

Area-array cameras and line-scan cameras are the two types of cameras that have distinct applications for measuring the invarients of a workpiece. These are normally high resolution CCD cameras used with microprocessor controllers. An area camera captures a scene by mapping it into a bidimensional pixel array and outputs the data frame-by-frame. Typically a camera with a 480 x 512 pixels resolution can scan the image at a rate of 30 frames per sec (11). The time interval required for the signal to travel from the camera to the processor is relatively high, and the volume of data supplied from each frame requires adequate processing time. This kind of camera is most suitable for

inspection of slow moving objects.

On the other hand, line-scan cameras employ linear arrays pixels and can scan line-by-line as fast as 100-40,000 scans per sec (11). Several cameras can be ganged to provide an extended viewing area. Line-scan cameras convey lower volumes of data than area cameras. The scan line of a line-scan camera must be positioned in such a way that it will capture adequate amounts of information concerning the variable of interest. The time interval between successive lines of data is determined by the level of illumination. This time is independent of the time taken by the data to travel from the camera to the processor. These properties make the line-scan camera suitable for scanning fast moving objects with high resolution requirements (5,6).

DETECTING AN EDGE OF AN OBJECT:

If we consider a black and white system, the image occupies two types of regions, object and background. The gray levels of an image can be a positive going or negative going strip depending upon the contrast between object and background. Figure 1 shows the gray level signal samples for four boards with a black background. The gray level intensity of an image depends on many

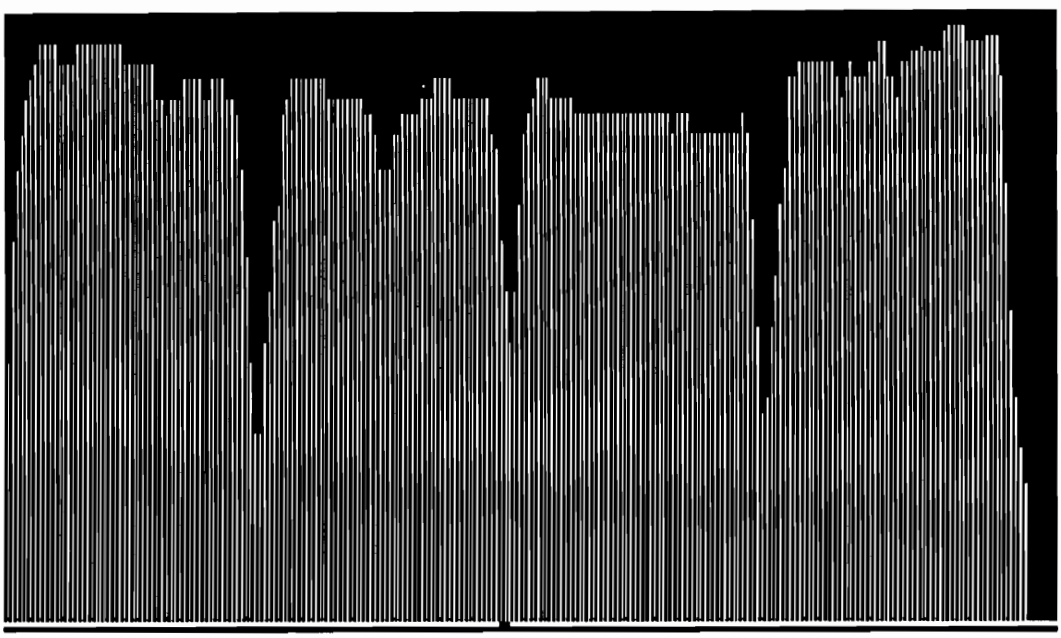


Figure 1. Gray level signal for four boards.

factors including size of the CCD array cells, exposure time, direction of the line segment connecting the surface point with the focal point, and surface reflection properties(5). In a realistic imaging system, the CCD array cells receive reflected radiance from the surface of the object and from its surrounding areas. Near the edge of the object the received reflectance from the object and the background have different reflection properties. This makes the gray level near the edge change gradually as shown in the Figure 1.

Thresholding is a technique that can be used to identify the edge point of an object. The general principle involved in selection of a threshold is to fix a gray level, say "t", above which the object is identified and below which is the background. If the level t depends only on the gray level at that point, then it is called a global threshold. If t depends on both gray level and some local properties, like average gray level of the neighborhood points, then it is called a local threshold. If it also depends on the coordinate position of the point, then it is known as a dynamic thresholding technique (10, 14, 12, 13). If the contrast between the object and the background is poor or the reflected light intensity is low, then the

threshold to distinquish the object from the background is difficult to fix. Again the optical properties of the surface of an object, for instance a wavy, reflective surface, can cause problems. Since these problems are intensity related, they can be reduced using appropriate lighting and filtering. The real objective of thresholding is to detect the presence and location of gray level changes in an image that represent an edge. Nonuniform thresholding(dynamic thresholding) is used in the places where uniform thresholding is not adequate. When using this technique, several assumptions are made regarding the input image. For instance, the intensities or gray levels in an image are assumed to be independent and identically distributed and an image is a two dimensional spatial process.

Various algorithms have been developed in recent years using adaptive filtering techniques. Mastin(7) discusses six different algorithms for a 512 x 512 pixel, 8-bit image. He uses a 7 x 7 pixel window to implement and evaluate the filtering algorithms. These six filters are median filtering, k-nearest neighbor averaging, gradient inverse weighted smoothing, sigma filtering, Lee additive and multiplicative filtering, and modified wallis filtering. These filters have good

smoothing capabilities in flat noise regions, are adaptabe to local contrast changes, have good computational speed, and are widely discussed in the literature in the past few years. Mean filtering is another one which uses a sliding window, running average approach. In this technique, the averaging sum of intensities updates for each incoming column and each out going column of the window as it slides over the data.

In a median filter, the intensity of the pixels within the window is replaced by the intensity of the median pixel to that particular window. The k-nearest neighbor averaging approach assumes the local window intensities are uncorrelated, and it replaces the sliding window's central pixel with the average of its k neighboring pixel values belong to the same population. In a gradient-inverse filter, a matrix of weighting coefficients, each element of which is the normalized gradient-inverse between the central pixel and its neighbor, is generated and applied to the local window's pixels (7). A Lee additive filter approximates the uncorrupted image mean and variance from the local corrupted image mean and variance. Here Lee assumes that the sample mean and variance of a pixel is equal to the local mean and variance of all the pixels within a

neighborhood. Lee's multiplicative filter is similar to the additive filter but is based on a sliding window mean, variance, and global noise estimate.

The sigma filtering technique (4,7) is based on averaging only pixels within two standard deviations from the intensity of the central pixel. The characteristics of this edge detection technique are its flexibility in setting the magnitude of edges to be detected and also its consistency in detecting edge because it is independent of the orientation of the object. This filter is based on the Gaussian probability distribution. The two sigma probability for a two-dimensional Gaussian distribution is .955, that is 95.5% of the random samples lie between plus and minus two sigma limits of the mean. This filter is suitable for detecting edges in our case. A modified wallis filter uses a statistical differencing operator to force a desired mean and standard deviation.

MEASURING AN INVARIANT OF AN OBJECT:

A geometric feature of a workpiece which does not change with the position and orientation of the piece during inspection is known as an invariant. For a rigid

workpiece, the area, width, length, and any permanent marks like defects are examples of invariants (2). Let us say we are interested in measuring the width of a board using a line-scan camera. Figure 2 shows the configuration for measuring the width of the board ABCD. Assuming the ideal case, the camera optical axis is perpendicular to the x-y horizontal axis. The board is placed on the x-y plane. The camera center is at 'O' with the coordinates (x1,y1,z1). Y1 is parallel to the width direction, z1 is the axis of the camera, and the image plane is parallel to the horizontal plane. If the focal length of the camera is 'F' and the distance between the board's surface and the len's center is (L - h), where h is the height of the board and L is the distance from the horizontal plane to the center of the lens (Figure 3), then the width of the board is calculated by the formula,

$$W = \frac{(L - h)}{F} W'$$

where W is the actual width of the board and W' is the measured width at the camera (5,6). If the camera is not in the ideal position, it is tilted and the y-coordinate of the camera center is facing in the

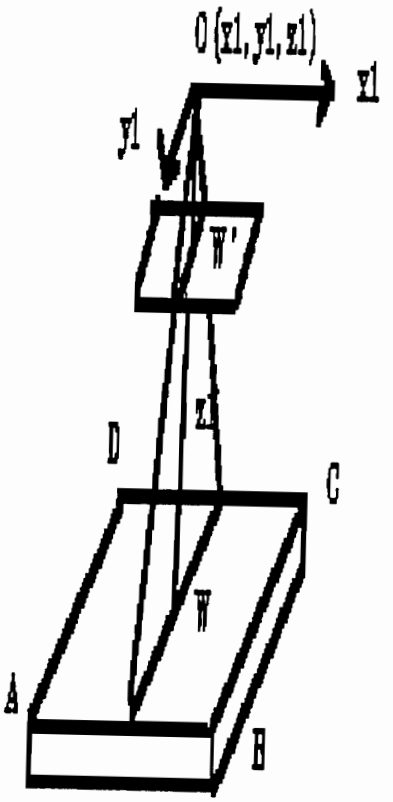


Figure 2. Measurement of board width.

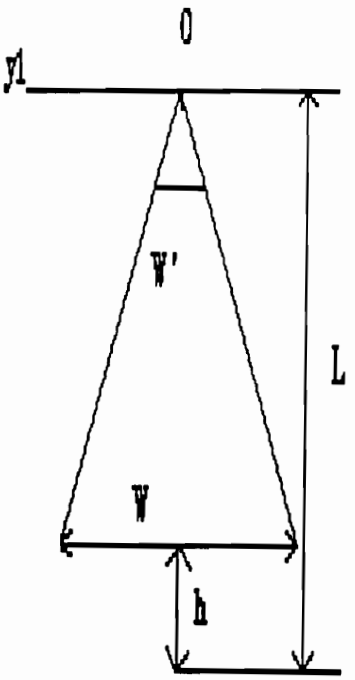


Figure 3. Measurement configuration.

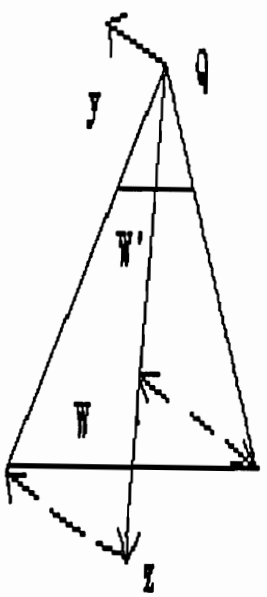


Figure 4. Tilted camera configuration.

direction of the row of CCD cells of the line camera and the x-direction is perpendicular to the y-z plane as shown in the Figure 4. In this case it is said that the configuration satisfies the right hand rule(6). To calculate the measured width of a board a new coordinate, say 'Q', is considered by rotation and translation of the coordinate system. Then the change in the coordinate systems O and Q is represented by a 3 x 3 rotation matrix and a translation vector. By manipulating the matrix, they derive an expression for W'. In general, as long as the object is within the view of the line camera, the measured width is independent of the x- or y-direction translation. But if only the tilt angle is considered, then the measured width is less than the true width. This is due to the fact that the distance in the field-of-view of the camera over the board surface increases with the tilt angle. If the aspect ratio, which is the ratio of the resolution of the x-axis to the resolution of the y-axis, of the camera is 1.0, then the measured width is independent of tilt angle. The rotation angle and the translation parameters are usually unknown; however, the system can be calibrated with an object of known size (6). Calibrating the camera system for certain positions with a single measurement is easier than going

to six degrees of freedom in translation and rotation of the camera system (6).

MEASUREMENT ERROR:

The measurement of an invariant using a digital vision system involves measurable variations. Chih-Shin Ho(2) talks about four kinds of errors involved in measuring an invariant: (1) motion of an object and the camera, (2) parallax, (3) poor contrast and low brightness, and (4) digitizing errors due to discrete sensors on an image plane. Since the pixel width is small, the image gets distorted when either the camera or object moves. During the movement of either one of them, the pixel width remains the same but the camera views more of the object surface. So for a real time operation, a correction factor for the increase in pixel length (x mm per scan) has to be added to the measurement. The farther the object is from the optical axis of the camera, the greater will be the exposed area of the object. If the vision analysis is done using this data, then the sides of the object can affect the shape, area, and geometric centroid of the image. This parallax error can be reduced by increasing the distance between the object and the camera. The third error

creates difficulties in determining the threshold level for the image and has already been discussed. If the image plane of the camera uses a discrete sensor array, then the image is transformed into discrete spaces called pixels. This digitizing process generates a digitizing error which can be reduced by increasing the spatial resolution. If the width of the pixel is 'q', then this digitizing error falls within +q/2 and -q/2. Increasing the resolution to reduce this problem increases the computation time for the vision analysis and possibly the hardware cost as well.

POWER CONSUMPTION IN THE SAWMILL INDUSTRY:

In a sawmill, power consumption data for a particular saw can be utilized for on-line monitoring of the process. As time goes on, the saw dulls and the accuracy of cutting decreases. That is to say, it removes more material as it cuts the log into boards. It has to work hard to cut through the log resulting in more power consumption. In general, the saw will be removed from the machine by a predefined time interval and sharpened for reuse. In this method, two kinds of problems may occur.

1. The saw teeth may have gone dull well before the

replacement time, consuming more power and cutting the board with less accuracy.

2. It may still be sharp enough to cut the board well within the limits.

These two problems can be solved by monitoring the status of the saw on a real-time basis. Continuosly collecting and monitoring the power consumption by the saw gives an idea of when it loses its sharpness.

CONTROL CHART:

A control chart is a graphical comparision of process performance data to computed "control limits" drawn as limit lines on the chart ..Juran(3). A control chart is an effective quality control system which rapidly detects the changes in the process variable being monitored. The primary use of a control chart is to trace "assignable causes" of variation in the process. Two kinds of process variations are detectable: (1) random causes solely due to chance, and (2) assignable causes due to specific "findable" causes (3).

An ideal process is expected to have only random causes. This means the process operates within allowable minimum possible amounts of variations. That

is to say the process is "in control". The random causes are unpredictable and also, can not be eliminated. The variation due to random causes is minimal and once the random variations are within the control limits, then the process should not be disturbed.

In contrast, the assignable causes are due to one or more known individual causes and result in large variations. These can be detected and the action necessary to eliminate them can be economically justified. The presence of this variation needs to be checked and corrective action taken.

A statistically control system does not necessarily mean the product meets specifications, while an out of control process might still be producing products which conform to specifications. These things may be true due to process misdirection, set averages, or too widely scattered data. This can be easily corrected. Setting up control charts for a particular characteristic of the process is a matter of judgment. This involves a few steps like identifying high priority characteristics, differentiating the process variables and conditions contributing to end product characteristics, choosing the characteristics which will provide the kind of data needed to diagnose the problem,

deciding on the implementation point in the process at which the test can be done, and selecting the type of control chart.

In constructing a control chart , first take a series of sample data from the process. During this process, keep track of changes in raw material, tools, etc. Compute the control limits for the trial data. Then chart the data for the running process against these control limits to determine the current status of the process. Figure 5 shows the process mean plotted as a function of sample numbers. There are two control limits called the upper control limit and the lower control limit. Control limits are fixed at +3 sigma or -3 sigma to the mean value. If a point falls outside these control limits, then the process is said to be out-of-statistical control.

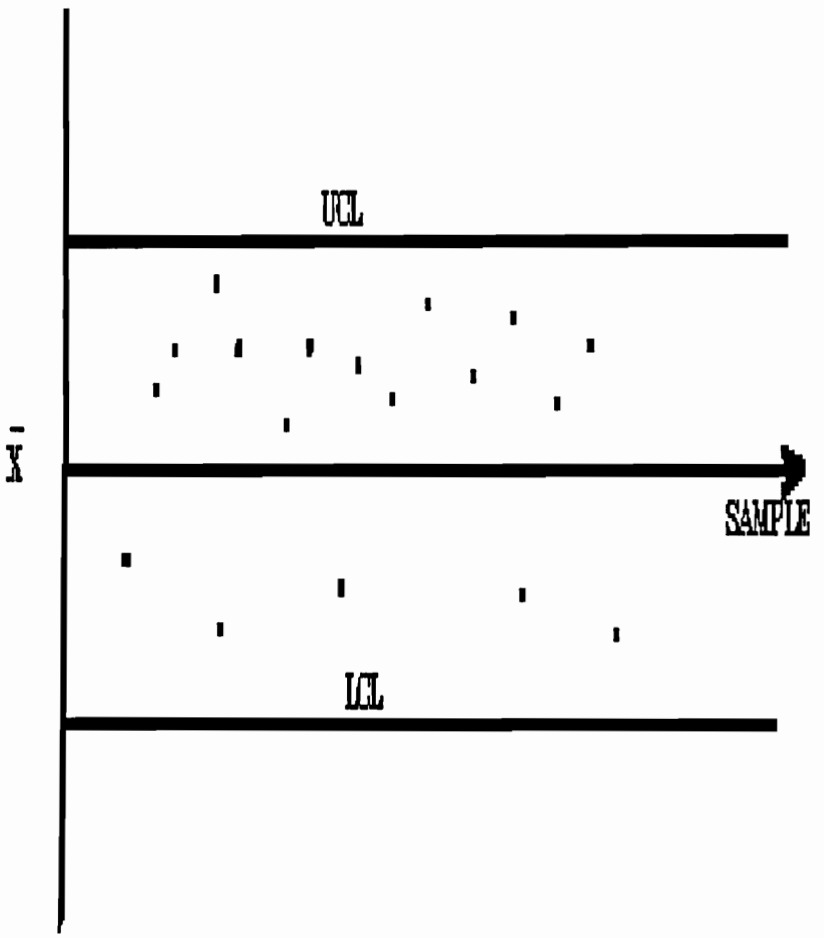


Figure 5. A sample control chart.

PROJECT OVERVIEW

The system used to reach the objectives stated in section two is shown in Figure 6. A line-scan camera from SAAB SYSTEM Inc, Seattle, Washington, is set up to scan the boards for their widths. The output analog gray level signal from the camera is digitized by a Tektronix 2430A digital oscilloscope. The digitized data in ASCII format are transferred to the computer through a GPIB(General Purpose Interface Bus). The received data are interpreted and widths of the boards are computed by the software system. The quality control system utilizes the computed widths and checks the average values for their statistical control status and reports back the status in different formats. A Power Master III AC multimeter collects the power line data and transfers the analog data in volts to the data acquisition system where the analog signal is converted into a digital signal and tranfered to the computer. The received data are checked for their statistical control status and the result is plotted or tabulated.

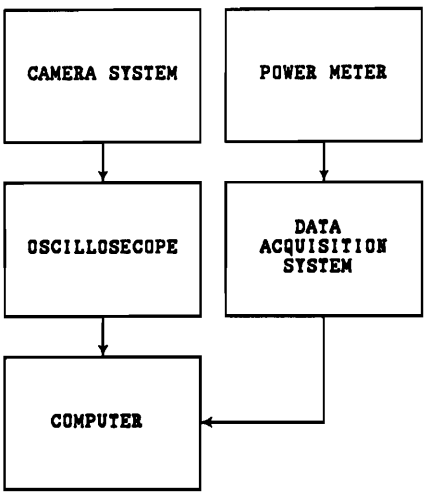


Figure 6. Block diagram for the system used in this research.

LINE-SCAN CAMERA:

The solid state camera system consists of a CCD linear array in its image plane. The array length is 1024 pixels. The focal length of the lens is 25 mm. It scans at a speed of around 1200 scans per second and the data are transferred line by line to the processing unit.

DIGITIZING SYSTEM:

A Tektronix 2430A digital oscilloscope is used to digitize the camera data. The scope has programmable features for automatic testing and measurement. It can be controlled remotely by means of a GPIB communication link. The scope is used in normal acquisition mode to acquire a single line of data. In normal mode, it acquires data to a horizontal length of 20.48 divisions and transmits this data as 1024 sample points through the GPIB port to a microcomputer. An IEEE 488 interface card from Capital Equipment Corporation is used to interface the scope and the computer. For a real-time installation, a high speed digitizing card would be normally used. However, the oscilloscope was used in this research because of its availability.

POWER METER:

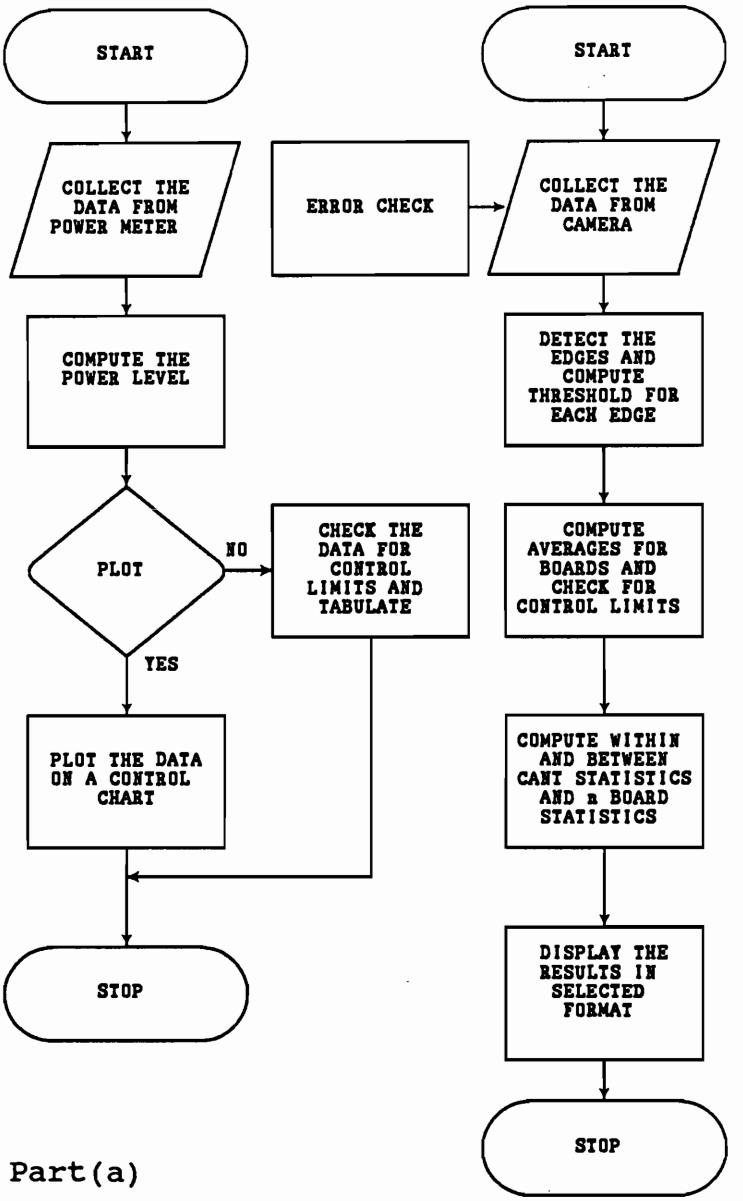
The Power Master III by Esterline-Angus has the capability to measure a multitude of power parameters using a microprocessor for industries to conduct plant power surveys and energy audits. Voltage inputs to be measured are coupled to the voltage multiplexer through line isolation transformers. All AC measurements are displayed by a liquid crystal display (LCD) located on the front panel. The input voltage measurement range is 60 - 600V rms. Input voltages are stored, selected and switched using a microprocessor controller. Autoranging amplifiers are used to fix the range between input and the output of the meter. Final scaled down analog output ranging from 0 to + or - 2.2V (with a load resistance 2.2 K-ohms) is transferred to the recorder output jack. This output signal is the proportional representation of the input signal displayed on the LCD indicator which is updated for every 2 seconds for single phase power supply and 3 seconds for three phase power supply.

DATA ACQUISITION SYSTEM:

A WB-FAI-B high speed interface card from OMEGA Engineering Inc., is used to digitize the analog output signal from the power meter. The system can be operated in menu driven mode or through our own program to suit our requirement using the system driver program. Voltage measurement range, resolution, and scan rate can be set for the application of use.

PROCEDURE

Flow Diagram 1 shows the overall quality control system. In Flow Diagram 1 Part(b), a single line of camera data is grabbed by the scope and digitized and then the data are logged into a file. The data can be plotted by a seperate program to view how the gray level signal spread corresponds to the acquired image. For deriving board width, the program reads the data, detects the edge of each board, and sets a threshold for each board. It computes the number of samples between edges that are above the threshold level for that particular board and then computes the average widths for each board. These averages are checked against control limits by the quality control program. Within and between cant statistics are computed and checked against control limits. The results are displayed in a selected format with out-of-control points highlighted. In Flow Diagram 1 Part(a), power meter data are converted to real levels. By option selection, the data are plotted on a control chart to check for the statistical control of the system or checked for control



Flow Diagram 1. Overview of quality control system.

limits and displayed in a table with out-of-control points highlighted. Appendix B contains all the program lists by the order of discussion in this section. Appendix A contains descriptions of all variables.

READING CAMERA DATA:

Program 'log.c' transfers data corresponding to one single acquisition to the computer and logs the data to a file. Flow Diagram 2 shows the steps involved in program 'log.c'. The file name for the data to be logged has to be given by command line argument. To acquire the data, the scope has to first be set for the following setup;

1. Trigger mode:
   - A/B TRIG : A Trigger & Auto.
   - TRIG POSITION : 1/2.(CHANGE FOR COVERAGE)
   - SLOPE : Positive.
2. Horizontal:
   - MODE : A.
   - A and B SEC/DIV : 50 us.
3. Vertical:
   - MODE : CH1,CH2 & YT.
4. Bandwidth:
   - BANDWIDTH : 20MHz.
   - SMOOTH : ON.
5. Acquire:
   - ACQUIRE : Normal.
6. CH1:

COUPLING/INVERT : DC.
VOLTS/DIV : 2V.

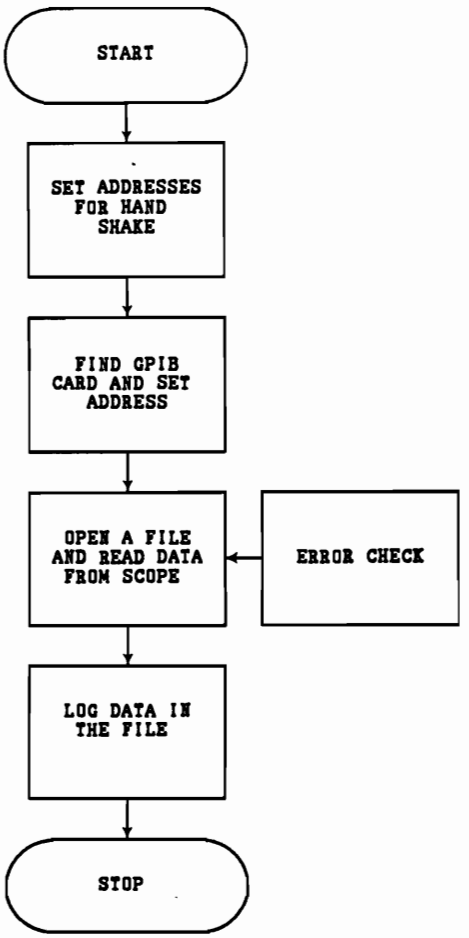
7. CH2:

COUPLING/INVERT : DC.
VOLTS/DIV : 1V.

8. OUTPUT:

SETUP : ADDR TO 17

The camera signal is fed to channel 1 and the synchronizing signal to channel 2. The analog signal from the camera is converted to a digital signal and displayed on the oscilloscope screen. The trigger level knob is used to stablize the signal on the screen. The program 'log.c' is run by entering 'log <file name>'. The program sets addresses to the <PC-488> card(CEC interface card to the GPIB), GPIB addresses to the IBM card, and GPIB addresses for the TEK scope. It also sets the GPIB bus level to zero. The GPIB card is found by the routine 'findgpib()', and if the address is not found it reports that the card was not found and exits. If the address is correctly found, then it initializes the GPIB card for the communication handshake. A file is opened to write the data and a message is sent to the scope asking it to send the 'curve', which corresponds to a single acquisition. Data corresponding to 20.48 horizontal divisions of the scope are sent as 1024 data samples. The signal levels correspond to intensity

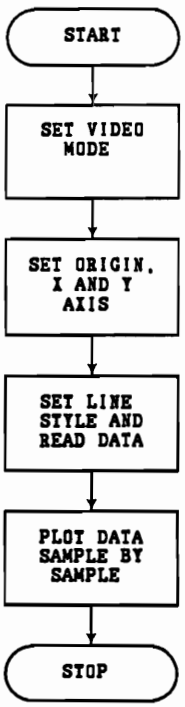


Flow Diagram 2. Reading the camera data.

levels of the signal on the image plane of the camera. The err_enter() routine checks for any handshake errors in the communication between the devices, scope, and computer. Samples are read one by one, the comma(,) is removed from between the sample values, the data are logged to the given file. If the program needs to be changed for any reason, it can be recompiled by linking it with 'gpibmsc.lib', which is available in the <PC-488> interface card programs.

PLOTTING DATA:

Flow Diagram 3 explains the program 'plot.c', which reads the data from a file given by the command line argument and plots it sample by sample on the display. If the name of the file to be read is not given, then the program terminates. To execute the program, enter 'plot <data filename>'. The data file is the output file of program 'log.c'. Before running the program to plot the graph, run 'MSHERC.COM' for hercules graphics. The '_graphics_mode()' routine gets the present video configration of the screen, sets it to the hercules video mode by _setvideomode (_HERCMONO), and configures the screen for hercules mode. The program can also run in other video modes just by changing



Flow Diagram 3. Plotting the camera data.

'_HERCMONO' in 'plot.c' and recompiling the program to any other required modes like EGA or VGA modes. The exact phrase to be used for various graphics display modes is given in the book 'Microsoft QuickC - C for Yourself' page 201(9). Newx() and newy() are two routines which are used to refix the screen for the number of pixels needed to suite our requirement in the x-axis and the y-axis. In 'plot.c' it is fixed for 1050 pixels in the x-axis and 100 pixels in the y-axis. This reconfiguration of the screen to our requirements gives room to fit all the sample points in a sweep and highlights the critical points to be analyzed. Subroutine '_setvieworg()' sets the new origin to our plot and that is set at half the height of the reconfigured screen. The coordinate point (0,50) becomes the origin. A vertical line and horizontal line bifurgate the screen. The area above the horizontal line represents the negative portion of the y_axis because the y-axis increments from the general origin coordinate(0,0) at the top left corner of the screen as it goes down to the bottom of the screen. In our set screen, now the bottom half is the positive y-axis.

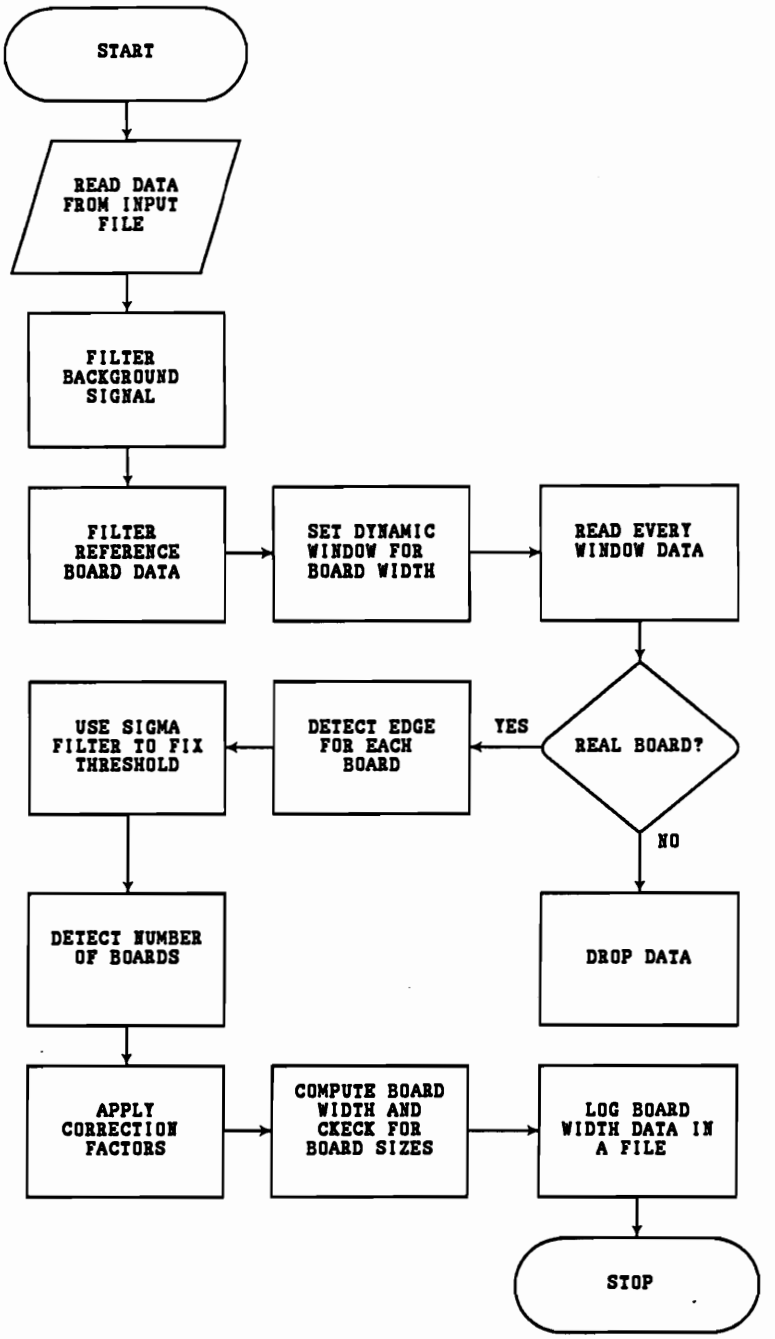
Subroutine '_setlinestyle()' sets the line style to be used for drawing the line between x-axis and the

data point. After setting the screen, the program opens a file, reads the data, and starts plotting sample by sample as it reads 'MAX' number of sample points. Subroutine '_moveto()' first locates the cursor at set origin (0,0) and draws a line from the first sample point to the origin. The height, which is in the y-axis, represents the value of the intensity level of the signal at that point. It then moves to next sample level and draws the line to the x_axis and so on. Subroutine '_lineto()', draws the line from the present position of the cursor to the set coordinate point. After plotting the data, subroutine 'end_program()' ends the video mode when any key from the keyboard is pressed. Subroutine '_displaycursor()' puts the cursor back in visible mode. The plot can be redirected to the printer using the Microsoft Software Corporation 'Paintbrush'(8). When Paintbrush is first started, it loads the Frieze program to memory. 'Frieze' remains memory resident when Paintbrush is exited. Program 'plot.c' is run to display the plot and SHIFT-PRINTSCREEN are pressed to display the Frieze menu. Selecting the save option from the menu will save the current screen with a file extension of '.pcx'. Paintbrush is then restarted, loaded with the saved file by selecting the load option from the file menu, and

printed by selecting the print option.

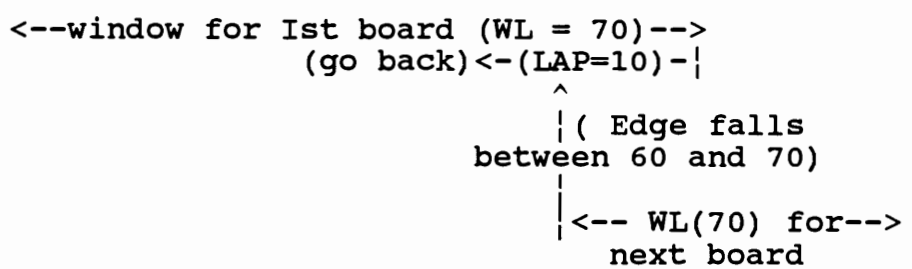
COMPUTING BOARD WIDTHS:

Program 'width.c' computes the width for each board from the camera signal. Flow Diagram 4 shows the structure of the program. To run the program, enter 'width < filename for input data> < filename for output data>'. Otherwise, execution of the program will be terminated. The input data file is the output data file of program 'log.c'. At the start of the program, the camera distance from the surface of the board has to be entered when the prompt to enter this data appears on the screen. The program reads the data and filters the samples corresponding to the background signal, which are signal levels up to intensity level 12. It resets the 'MAX' maximum number of samples and sets 'WN' number of boards that can be detected from the available data for the given window size 'WL'. WL is set for 70 samples. WN can be set up to 15 boards. A sigma filter, which was previously discussed, is used to fix the threshold for each board signal detection. Window size 'SW' is fixed for the sigma filter to 7 samples. The
first SW samples are taken to fix the mean 'ave' and 2



Flow Diagram 4. Computing board widths.

sigma points 'ul[row] & ll[row]' for the sigma filter. The mean is checked for real boards. Since the background signal is filtered up to an intensity level of 12, the computed mean for the sigma filter should be above 13 for the boards. If it is below this level, it is not a real board. Then 'shift' fixes WL number of samples for one window either from the starting of data or from the detected edge of a board, depending upon the current running status of the program. It then goes back to the 'LAP' number of sample and searches for a valley(minimum) point in the data to fix the edge for that board. Lap is fixed at 10 samples. Then WL for the next board is fixed from the next sample of this edge. From there the sigma filter parameters and other parameters for the next board are computed. The windows are moved dynamically depending upon the detected edge point of each previous board as shown here,



In case the LAP number of samples does not compute a valley because of the presence of an elevated

background signal, then the routine fixes a width to around 62 samples. After detecting the edges, the sigma filter computes the number of samples falling between ul[row] and ll[row] within the window SW. The average of these samples is the threshold for that particular board. By taking into consideration the effect of light reflection properties, some smoothing over the threshold is done by subtracting a constant value from the different threshold levels. After this smoothing operation, the number of samples falling between two edges of a board above the threshold are counted towards the presence of a board. This routine is continued until all the 'k' number of boards are detected and the number of samples 'array[k]' for each board are computed.

To compute the width of each board, the sample width 'sam_len' at the camera lens and some correction factors for scope error 'cor_fac' and quantization error 'det_err' for the sampled data should be calculated.

$$\text{sam\_len} = \frac{\text{CFW} \times \text{FL} \times \text{STW}}{\text{CDFC} \times \text{STBS} \times \text{SW}}$$

where CFW = Camera view field width

FL = Focal length

STW = scope transmission width

CDFC = Camera distance from conveyor

STBS = # of samples transmitted by scope

SW = scope width corresponding to field width

that is,

$$\text{sam\_len} = \frac{24 \times 25 \times 20.48}{42 \times 1024 \times 16.8}$$

The edge of the board does not need to fall at the beginning or end of one sample; it can fall any place within a sample. This is a quantization error in the analog to digital conversion of the camera signal. This error generally falls within + or - 1/2 sample. If 'sf' is the distance from the camera to the surface of the board, then

$$\text{det\_err} = \frac{\text{sf} \times \text{sam\_lem}}{2 \times \text{focal length(25)}}$$

The measurement by the scope in relation to the measurement at camera lens induces some error. This error is determined by doing the reverse calculation. A known board width of 1.48 inches and camera distance of 38.5 inches gives a measured width 0.961038961mm (computed by 1.48 x 25 / 38.5) at the camera. It covers

approximately 47 samples in the analog to digital conversion by the scope. Calculating in reverse (sam_len x 55) gives 0.935385mm. The difference is 0.025653961mm. So the error per sample is 0.000466435mm.

cor_fac = sf x (array[k] +or- .5) x 0.000466435inches.

The width of the board is computed by,

Width = (sf x sam_len x array[k])/ focal length.

Then the cor_fac is added. The det_err is added or subtracted depending upon the number of samples computed for each board based on a trial and error method of confirmation. If the number of samples detected for a board excedes 60, then it is considered to be too wide a board. If it is less then 50, then it is a narrow board. This narrow board result can be due to really having a narrow board or to the presence of a knot, broken edge or wedge. Similarly the wider board may result from a truly wide board or from the blacking of edges between boards by a piece of wood or anything else. Finally, computed board widths are logged into a given file.

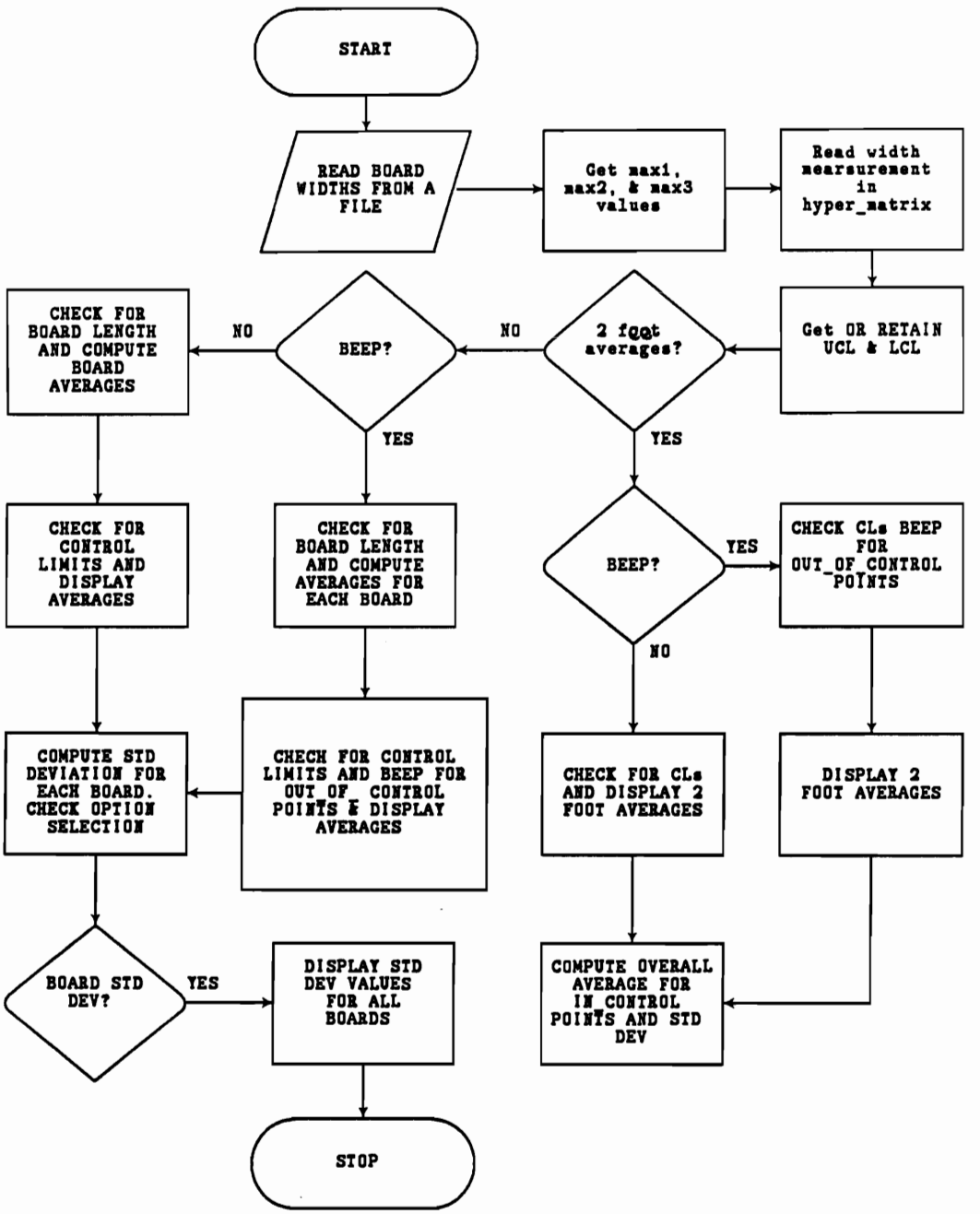
QUALITY CONTROL PROGRAM:

A quality control check on the computed board width is done by program 'q_con.c' and the results are displayed. To execute the program, enter 'q_con < input file name >'. The input file is the output file of program 'width.c' which contains board widths. If the input data file name is not given, then the program will be terminated. Flow Diagrams 5 to 9 give the steps involved in the whole program. The number of cants 'max1', number of readings per board 'max2' and number of boards in a cant 'max3' must be entered through standard keyboard input when the statement appears on the screen. Enter initial control limit values 'UCL' and 'LCL' for the first run and retain or change them in later runs. The program reads board widths into a matrix called 'hyper_matrix' for each cant. Assuming readings are taken for every 2 feet length of a cant, the widths will be ckecked against the given control limits by the routine tf_ave(). Entering '2' will ask whether the 'BEEP' is required for out-of-control points or not. Accordingly, the widths will be checked and displayed with out-of-control points highlighted. The average and standard deviation of the group of boards falling within the control limits are computed and

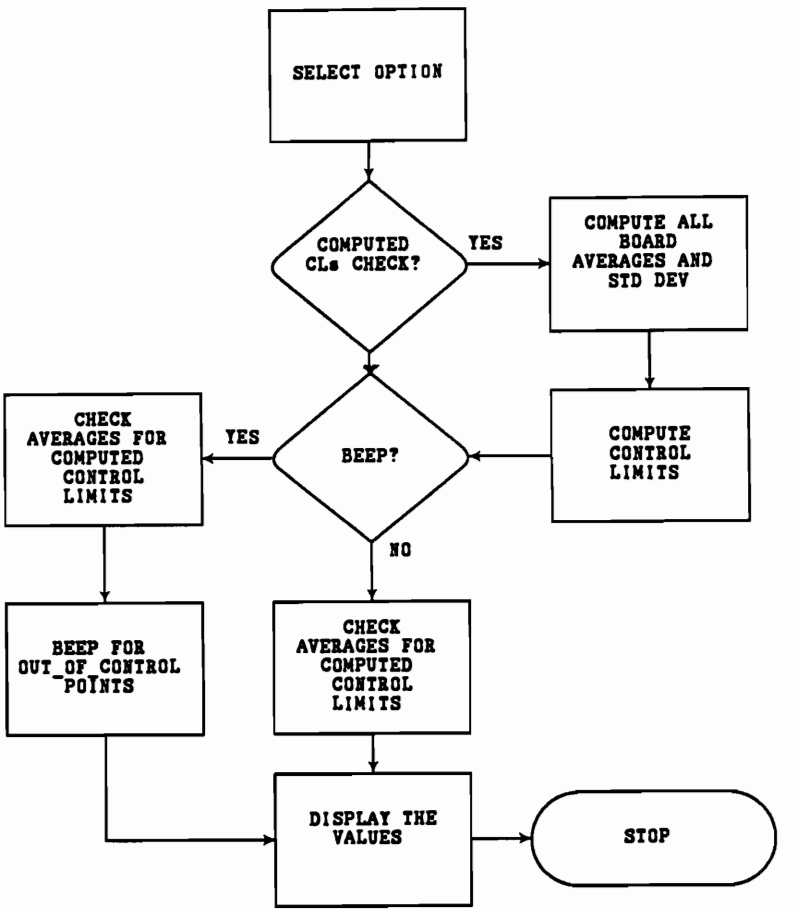
displayed.

Board averages, 'average[][]', for all boards are computed by checking values of each computed width that fall between 1.4 inches and 1.5 inches with the routine board_ave(). This is done to eliminate the erroneous readings due to knots, wedging, or any other defects. If all the readings of a board lie outside these checking limits, the average is still computed. Then the control limits check is done. By entering the option for 'BEEP", the result is displayed with the out-of-control points highlighted. Before computing the board average, the number of readings in each board is checked to see if it is more than 18, because the maximum board length is 26 feet. The routine board_dev() computes the standard deviation 'Sc[][]' of each board following the same strategy as the average computation. These steps are shown in Flow Diagram 5.

The selection menu is displayed to select the kind of result needed to check the process status. By pressing '1', board standard deviations are displayed and the execution is completed. By pressing '2', the routine 'ccl_ck()' is executed. Flow Diagram 6 shows the steps involved. Here the control limits for all the boards in the input file are computed with 3 sigma limits and the board averages are checked against these



Flow Diagram 5. Computing board averages and standard deviations and checking for control limits.



Flow Diagram 6. Computing control limits and checking averages.

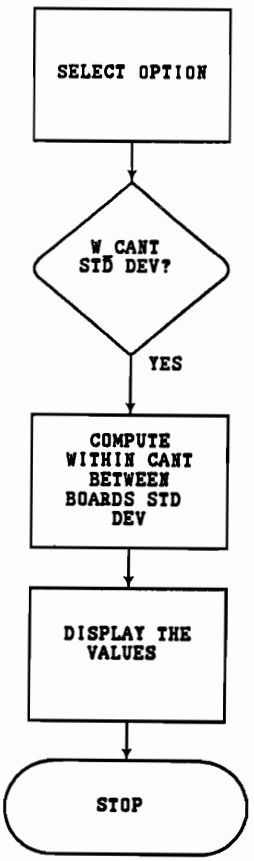
computed control limits. The result is displayed by option with or without the 'BEEP', and the program execution ends.

Flow Diagram 7 shows selecting '3' implements the routine 'w_cant' which computes the standard deviation for the boards within a cant for each cant and then terminates the program execution. The formula for computing within cant standard deviation is,

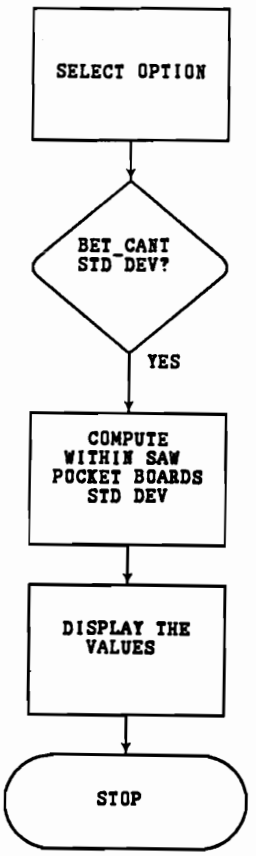
$$Swc = SQRT\left(S_{\bar{X}}^{2} - Sw^{2}/n \right)$$

Where $S(\bar{X})$ is the standard deviation computed from the averages of the boards present in the cant. Sw is the standard deviation of the boards and n is number of measurements per board (1).

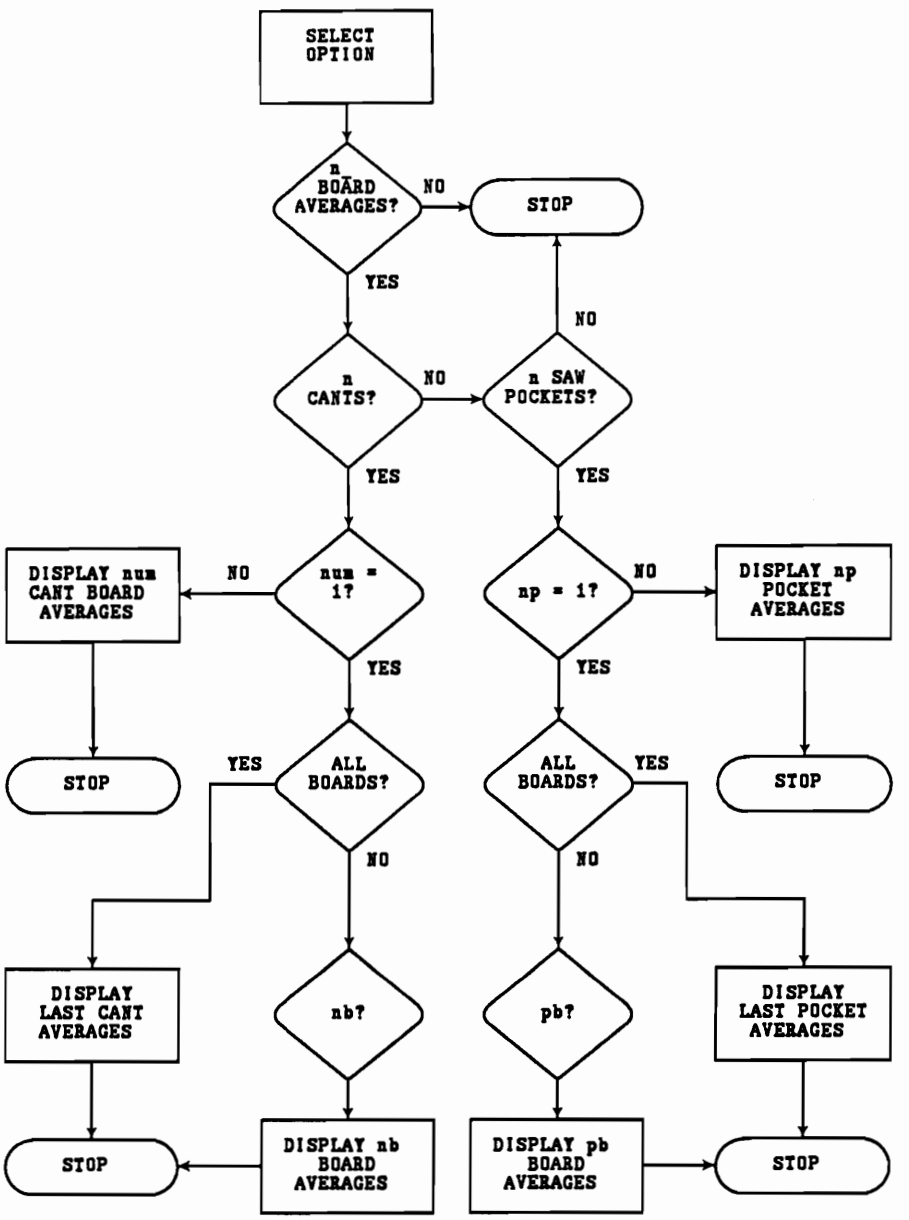
Flow Diagram 8 shows routine 'bet_cant()' which computes the between boards standard deviation for each saw pocket. The option is '4' and the program execution terminates at the end of the routine. The formula for computing this standard deviation is similar to the within cant between board standard deviations. Flow Diagram 9 shows that by the selection of '5', routine 'n_board()' is executed. By a series of questions, the routine displays any results wanted. The results are either the averages of the last cant or last pocket, last few cants or pockets, or last few board averages



Flow Diagram 7. Computing within cant between board standard deviations.



Flow Diagram 8. Computing within saw pocket board standard deviations.



Flow Diagram 9. Computing n board averages.

from the last cant or pocket. Then the program execution ends. If none of these results are wanted, then pressing any key other then 1 to 5 will end the program after displaying the board averages. The maximum number of cants that can be run is 10, the maximum number of readings per board is 20, and the maximum number of boards in a cant is 15.

Program 'bis.c' has the routines needed to set the screen for display options. Routine printa() sets the attribute to highlight the data displayed. Routine pos() positions the cursor for the current column position. Routine row() positions the cursor for the current row position. Routine check() checks the cursor's current position.

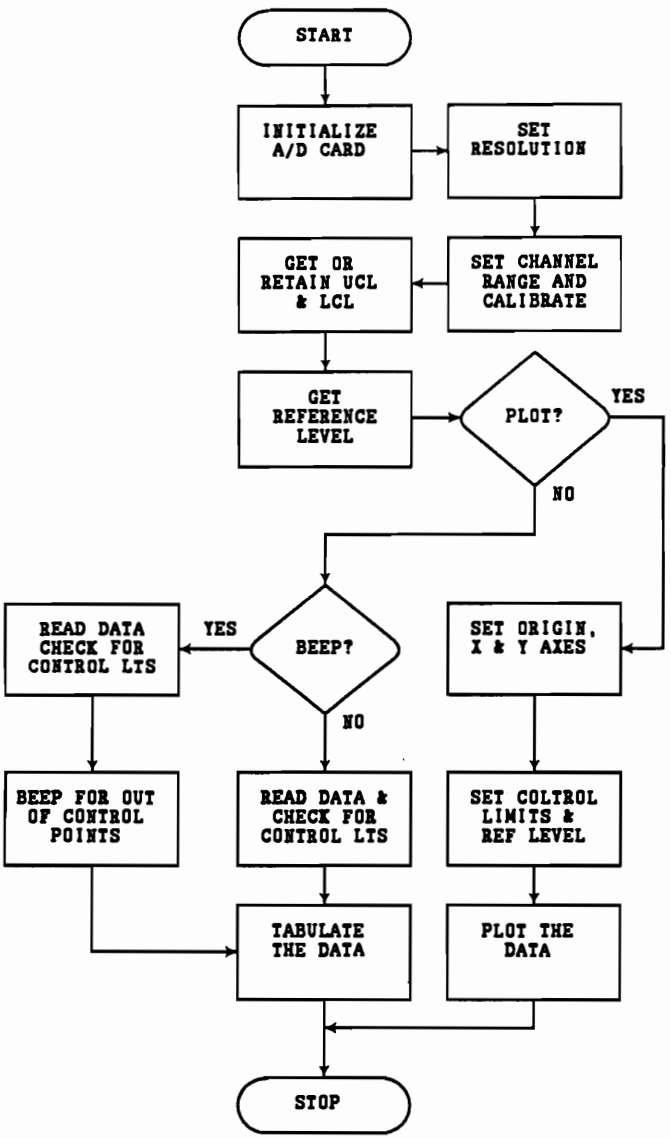
POWER MEASUREMENT:

Program 'power.c' reads the power meter data through the A/D card, computes the power value, and plots the data or displays it in a tabular form. The program uses the driver program 'ADRIVE.COM' for setup. Before running 'power.c' by entering 'power', run 'ADRIVE' to install the driver for the A/D card. To locate the data acquisition card, run 'FIND.EXE' by entering 'FIND -D -C'. This locates the card, loads the

calibration numbers from the disk, and tests and calibrates the card.

Program 'power.c' is shown in Flow Diagram 10. It gets the hardware setup and reads the calibration files. It also checks whether the driver and the analog card are installed; if not, it terminates the execution by an error message. Similarly, it checks for the analog card selection by running the program 'FIND.EXE', and validates the calibration numbers and buffer size for the channels. It then sets the analog channel resolution to 12 which is the low noise mode operation. This sets a resolution of 18 bits and the scan rate for data is 2500hz. The analog channel range is set to 4, which corresponds to + or - 250mv. This completes the initialization and setup of the A/D card.

The program needs an UCL, LCL and reference level to set the plot. After entering these data through the keyboard, an option can be selected to plot the data or tabulate it. This is done by subroutine 'display_data()'. If the plot option is selected, then it sets the origin and x and y screen size as explained earlier in the discussion of program 'plot.c' and reads the samples one by one and plots them. If the end of screen is reached, it clears the screen and continues the plot. This plot is a control chart, so any outlier



Flow Diagram 10. Power measurement flowchart.

can be identified just by looking at the chart.

If the plot option is not selected, then routine 'tabulate()' will tabulate data with or without the 'BEEP' signal for outliers after checking against the control limits. The connected load is 993 ohm for the output of the meter, so the input signal to A/D card is + or - 993mv. The peak value corresponds to an input of 600V AC(rms) to the meter, so the displaying data value in our program has to be changed to the actual scale. The value read for each sample is 'analog[0]'. Then the real scale value is,

Input = (600 * analog[0])/ (.993 * 1.4142)

The maximum samples to read are 'MAX_SIZE', which is defined as 150. This can be changed to monitor the data continuously by modifing the read statement in the routine to make it a while loop. If the program needs to be recompiled, then link it with 'MS_CALL.OBJ' and 'BASFUNCS.LIB'. These files are available with the data acquisition programs.

RESULTS AND DISCUSSION

The software technique developed in this research involves five steps. The first step is logging the data from the camera. Next is identifying the image by plotting the data. Then the data are analyzed and the widths computed for individual boards present in the acquired data. Finally, the averages of the boards are computed and checked against the control limits. A number of programs were developed to implement these steps and tested repeatedly for their results. This particular section of the discussion contains a set of results for each program and the specific setup arrangements used to get those results.

LOGGING CAMERA SIGNAL:

An example of the resulting logged camera data using the program 'log.c' is shown in Table 1. The data correspond to a single scope sweep and are in ASCII. The 1024 data points include a portion of the retrace path of the scanning beam, the boards, and the

---

Table 1. Example of logged camera data for clear boards.

---

3 3 3 3 3 3 3 3 3 3 3 2 2 2 3 2 2 2 2 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 2 3 3 3 3 3 3 3 3 3
3 3 3 3 1 1 0 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 3
3 3 3 3 3 3 3 3 3 2 1 1 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 2
2 2 2 3 2 2 2 3 3 3 2 3 2 3 5 7 8 9 9 10 10 10 10 10 10
10 10 10 10 8 7 7 9 9 9 9 9 9 9 9 10 10 10 11 11 11 11 11
12 11 11 11 11 11 11 10 10 10 10 9 9 9 9 9 9 10 10 10 10
10 9 9 9 9 9 9 9 9 10 10 10 10 10 10 10 10 10 10 10 10 10
10 8 8 8 9 9 9 10 10 11 11 11 11 11 11 11 11 11 11 11 11 10
10 10 11 11 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 9
10 10 10 10 10 10 9 9 9 7 7 7 9 9 10 10 10 10 10 10 10
10 9 9 9 10 10 10 10 10 10 10 11 11 11 11 11 10 10 10 10
10 10 10 10 10 10 10 10 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
9 10 9 9 9 7 7 7 8 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 9 9 10 10 11 10 10 10 10 9 9 9 9 9 9 9 9 9
8 8 8 9 9 9 9 9 9 9 9 9 10 9 9 9 10 10 10 10 10 10 10 9
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 10 10 10 10
11 11 11 10 10 10 11 11 11 11 11 9 9 8 10 10 11 10 11 11
11 11 11 12 12 12 12 12 12 12 12 12 12 12 11 11 11 11 11
11 11 11 10 10 10 10 10 10 10 10 10 10 10 9 9 9 9 9 10 8
8 9 ---> Start of first board --> (13) 17 25 35 45 51
55 57 58 59 59 59 60 61 62 62 62 62 62 63 63 64 63 63 62
62 62 62 62 62 62 62 63 63 64 64 64 64 64 64 64 63 62 61
61 59 59 59 61 61 62 62 62 62 62 60 56 49 37 28 -->
Second edge detected --> (24) 29 35 43 49 53 56 57 58
58 58 58 58 58 58 58 58 58 58 58 58 59 59 59 59 59 59 58
58 58 58 58 59 57 57 56 58 58 59 59 59 59 59 58 58 59 53
52 51 56 57 58 58 58 57 55 52 46 38 29 23 20 --> Third
edge detected --> (19) 19 21 25 32 37 41 43 45 45 46
46 46 47 48 49 50 50 50 50 50 49 49 49 49 50 50 50 49 49
49 50 50 52 54 55 55 55 54 54 54 55 56 56 56 52 51 51 55
56 57 56 56 57 57 56 55 54 53 51 46 39 31 25 22 -->
Fourth edge detected --> (21) 23 29 37 46 52 56 59 61
63 64 65 65 66 66 65 64 63 63 63 61 61 60 60 59 59 59 60
54 54 54 60 62 63 63 63 63 63 63 64 64 63 63 62 65 65 66
65 65 65 65 65 65 65 65 65 64 63 60 53 43 34 --> Fifth
edge detected --> (29) 29 32 38 44 49 52 54 55 56 56
56 56 57 56 57 57 58 58 58 57 57 57 57 57 57 57 53 53 52
56 56 57 57 57 58 57 57 57 57 57 57 57 57 57 57 57 57 58

Table 1 continued

58 58 58 58 57 56 55 55 53 47 40 31 26 23 --> Sixth edge
detected --> (22) 22 24 27 34 42 49 53 56 58 60 62 57
57 57 63 64 66 68 69 69 69 68 68 68 66 65 65 66 66 66 66
66 66 66 66 66 67 67 67 67 67 67 66 66 65 65 64 64 63 62
60 57 52 46 41 33 27 21 17 15 13 --> Seventh edge
detected (12) 12 11 11 11 11 11 11 11 11 11 10 9 9 10
10 10 10 10 10 10 10 10 10 9 9 9 9 9 9 9 9 9 9 9 9 9 9
9 9 9 9 9 9 9 10 9 9 9 9 9 9 9 9 9 9 9 9 8 8 8 9 9 9 9 9
9 9 9 9 9 8 7 7 9 9 10 10 10 10 10 10 11 11 11 11 11 11
11 11 11 12 12 12 11 10 8 8 7 7 6 7 6 6 6 6 6 6 6 5 5 4
4 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 3 3 3 3 3 3 2

---

background (here the background is the conveyer belt). The points of low intensity level values of 3 or 4 correspond to the retrace path, while the data values 7 through 12 correspond to the conveyer. The intensity levels that correspond to the presence of a board fall in the twenties and above, depending upon the color of different species of wood. The start of the camera field- of-view falls inside the conveyer belt in the present setup and extends a length of 24 inches. There are six real boards and the details are marked in the list.

PLOTING CAMERA DATA POINTS:

The data given in Table 1 were plotted in Figure 7 using the program 'fplot.c'. The program is the same as 'plot.c' except that it has a filter to filter out the background signal, and it expands the x-axis scale to plot the real board data. There are six boards shown. The y-axis represents the intensity level and x-axis represents number of samples. Figure 8 is a plot for the same set of boards, but this time the data were captured at a different place along the length of the cant. The plot shows the presence of a knot in the right side edge of the fourth board and another knot at

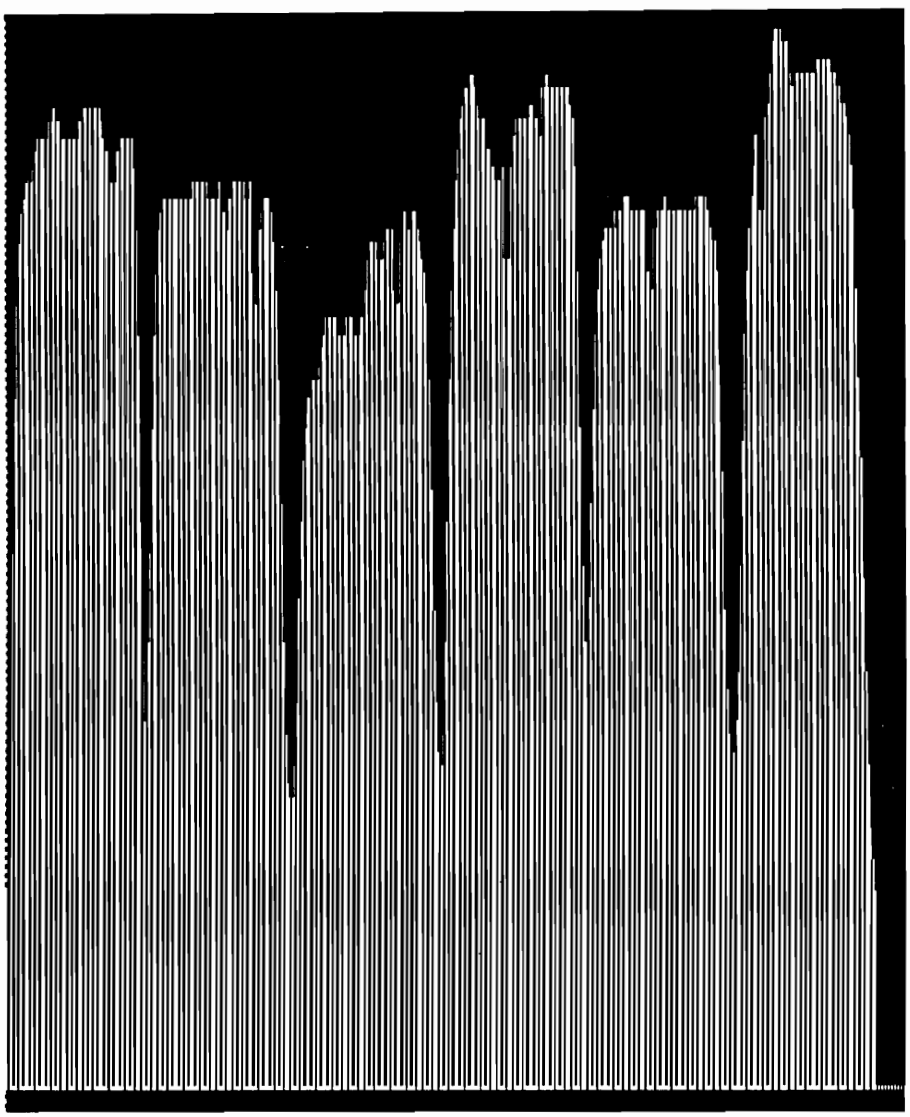


Figure 7. Plot of a six board cant.

Table 2. Program results for data in Table 1.

Number of sample points corresponding to the real image: 374

Number of boards present : 6

Detected edge points:

Intensity level	Starting sample point for next board
24.000000	62
19.000000	124
21.000000	189
29.000000	251
22.000000	314
0.000000	376

Threshold levels:

COMPUTED	SMOOTHED
34.428571	31.428571
50.142857	40.142857
34.857143	31.857143
48.571429	43.571429
46.285714	41.285714
40.714286	35.714287

Number of boards detected, k = 6

Sample width for the present setup : .017007

BOARD #	# OF SAMPLES	COMPUTED WIDTH
		-- inches--
1	56	1.479459
2	55	1.466130
3	58	1.532772
4	56	1.479459
5	55	1.466130
6	51	1.372831

the left edge of the second board.

COMPUTING BOARD WIDTHS:

Data in Table 1 are used to compute the board widths. Program 'width.c' reads the sample points from the input file and filters the background signal points up to an intensity level of 12. The remaining sample points belong to the image captured by the camera for the boards in one single scan. There are 374 sample points. The results are shown in Table 2.

Using a dynamic window, the first 70 (WL) sample points are taken from the filtered data points. Among the 70 points, the first 7 (SW) points are used to derive the sigma filter parameters. If the sigma filter average is above 14, then it is considered to be a real board; otherwise, that set of data is dropped. The last 10 (LAP) points in the 70 point window go back into the window to fix the edge point for that particular board.

The results in Table 2 show that the last edge point has an intensity level of zero, which is not true. Once the sixth edge detection is over, the window for the next board is fixed using the next 70 sample points. In reality, there may not be 70 points available because the background signal has been filtered out. So, the

Table 3. Computed and measured values for clear boards.

MEASURED VALUE	COMPUTED VALUE
-- inches --	-- inches --
1.4805	1.479459
1.470	1.466130
1.5025	1.532772
1.4805	1.479459
1.4720	1.466130
1.4480	1.372831

window is automatically filled with zero's for non-available points. When the last edge is detected, the signal intensity level drops sample after sample. If this happens for more than 8 sample points, then it is assumed to be the end of the image. In this case, the edge point is fixed at 62 samples. In the example data in Table 1, the 62nd sample has a value of 12. This is filtered out as a background signal and zero's are filled in to complete the 70 point window. When the routine goes to fix the edge at the 62nd sample, it reads a zero. This will not affect the results in anyway in later computations.

The sigma filter parameters for each board are computed by systematically moving the windows to every single board in the captured image. For each board, the average intensity value is computed for all the samples falling within + or - 2 sigma deviations of the sigma filter. These are the dynamically fixed threshold values for the boards.

There are cases in which the data collected may need some kind of smoothing on the intensity levels. This is done by subtracting a constant dc value from the threshold signal to fix new levels, but this setting may or may not work for other sets of readings. Most of the time the smoothing is necessary because of the fact that

the threshold value falls in a very high range. This results in a significant number of data points being ignored when accounting for the presence of the object. The best way to eliminate the error is to position the camera to get an optimum setup. The second important way is to adjust the intensity level of the lights for uniform distribution over the whole field-of-view.

For each board, the number of sample points with values above the threshold value is used to account for the presence of the board. The actual number of boards present in the captured image and the number of samples for each board are then computed. The sample width, 'sam_len', is computed for the given distance of the camera from the board surface. In this research, the distance was 38.5 inches, and the field-of-view for the camera was 24 inches. Then the width of each board was computed. Finally, the correction factors for quantization error and scaling error induced by the scope were added to the computed width of the board. The computed board widths were logged into a specified file.

All the results for the camera data in Table 1 are listed in Table 2. The widths of the boards were measured using a digital caliper at approximately the same place where the image was captured across the cant.

The measured values along with the computed values are shown in Table 3. The differences between these two sets of values are due to approximations involved in the computation, light intensity, and camera position. The approximations result from converting pixel values into sample values with respect to scope and correction factors. The sample width of 0.017007 is wide enough to induce a large variation between these two readings.

Images were captured in different places along the length of the cant. Figure 8 is the plot for the image captured with a knot on the right side of the fourth board and a knot on the left side of the second board. The results are shown in Table 4. The results show a wide variation between the caliper measured values and computed values. This is due to the fact that the smoothing strategy used for the data in Table 1 does not work efficiently for this set of data because of the nature of the variations in the intensity levels. The presence of knots does not affect the results because even though the knots are dark enough to be distinquished, the threshold level is sufficient to nullify the effect.

Figure 9 shows a plot with the first board having a broken dark knot at the left edge of the board. The results are shown in Table 5. The threshold level is

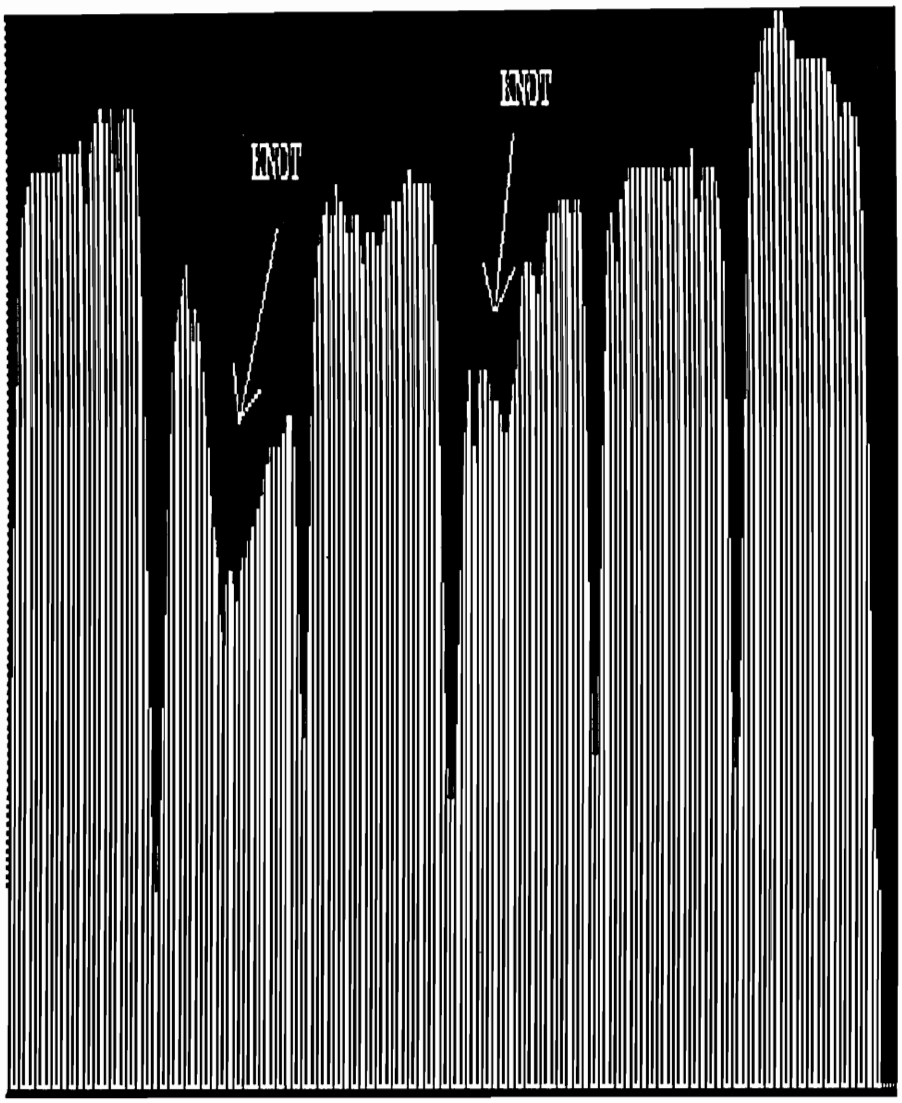


Figure 8. Plot of a six board cant with knots near edges.

Table 4. Results for boards with knots.

MEASURED VALUE	COMPUTED VALUE
-- inches --	-- inches --
1.4830	1.532772
1.4680	1.479459
1.4840	1.479459
1.4700	1.479459
1.4685	1.452802
1.4635	1.399488

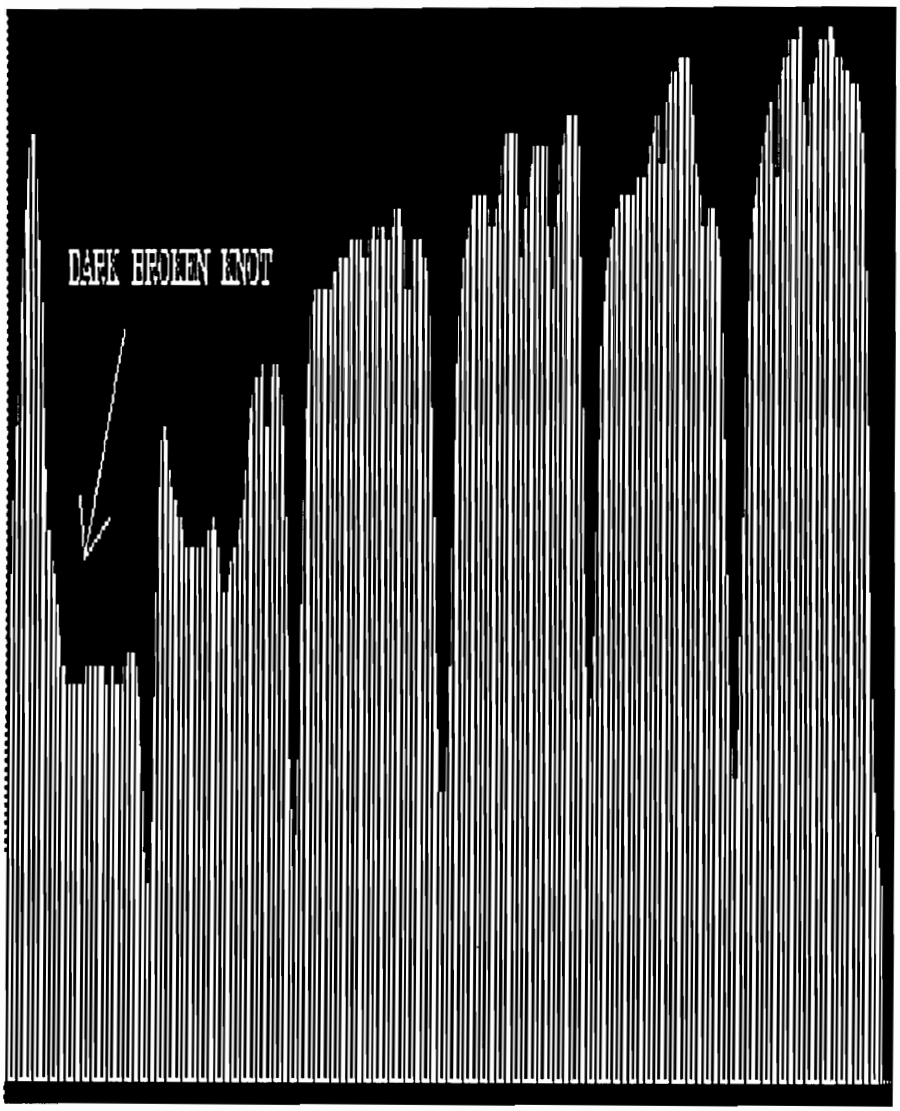


Figure 9. Plot of a six board cant with a broken dark knot in the first board.

Table 5. Results for a board with a dark broken knot.

MEASURED VALUE -- inches --	# OF SAMPLES	COMPUTED VALUE -- inches --
Board 1 is narrow or knot or wide wane		
1.4885	21	0.559795
1.4775	56	1.479459
1.4845	57	1.506116
1.4740	58	1.532772
1.4745	55	1.466130
1.4475	53	1.426145

fixed by the sigma filter by considering the first 7 sample points. The plot shows the sample points at the beginning of the board are at a considerably higher level than the end of the board, so the threshold is set at a high level. The computed threshold level is 35.428571 and the smoothing is done by subtracting 5, thus resulting in a threshold level of 30.428571. The knot is so dark that the intensity levels for the samples on the knot fall between 20 and 30. Therefore, these points do not count towards the presence of the object, which is the reason the results show the board as being narrow. Table 5 shows the variations between measured and computed values. The reasons for the discrepancies are the same as in the previous case.

Figure 10 is the plot for an image captured at a location which has a bright broken knot on the right side of the sixth board and the edge is covered between the first and second boards. The results are given in Table 6. The sample points corresponding to the knot fall at the starting point of the board, so the threshold is fixed while taking these points into consideration. Therefore, the result is not affected by the knot.

A piece of wood is blocking the edges between boards one and two. The measured value for the first

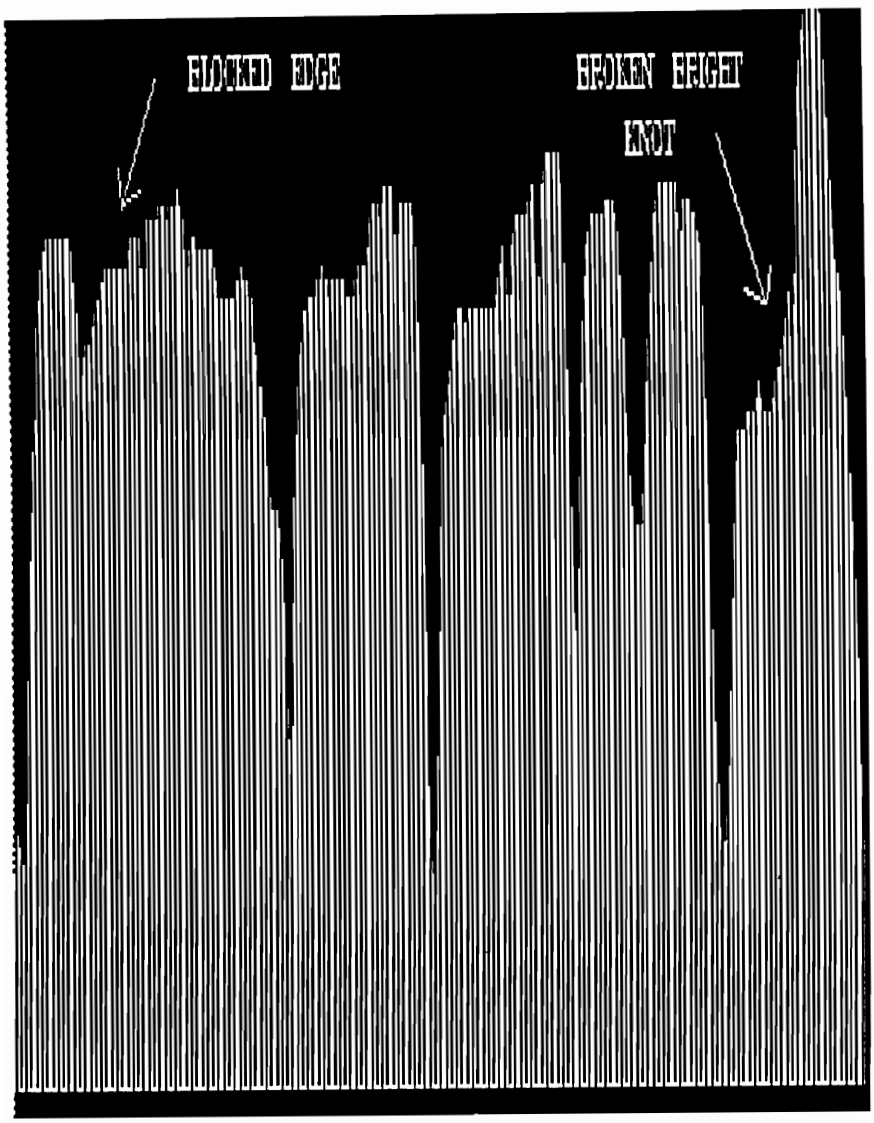


Figure 10. Plot of a six board cant. The edges between the first two boards are blocked with a piece of wood.

Table 6. Result for a light colored broken knot.

MEASURED VALUE	COMPUTED VALUE
-- inches --	-- inches --
1.1640	1.532772
Board 2 is narrow or knot or wider wane	
1.4680	1.226216
1.4950	1.452802
1.4815	1.532772
Board 5 is narrow or knot or wider wane	
1.4760	1.252875
1.4410	1.426145

board is narrow because it has wide wane on the outside edge. However, this effect is not shown in the computed value due to the presence of the piece of wood blocking the next edge. Since there is no edge point between boards one and two, the 70 point window with its edge detecting 10 point window automatically fixes the edge somewhere between 60 and 70 sample points. Therefore, that makes the next board too narrow. Board 5 is shown as being narrow in the computed result. The dark knot present at the center of the board pulled down 7 sample points below the threshold.

Figure 11 shows the plot for a reference cant with 12 boards. Now the camera distance from the object has been changed to 40 inches. The reference piece was machined from a block of plastic and was painted to give it a wood appearance. Because the field-of-view increases as the distance from the object to camera increases, the error computation routine was changed to match the height. The program changes are listed in Appendix B.

The results are shown in Table 7. Also shown are the number of samples counted for the computation. The height difference between the six board cant and the 12 board cant is 1.5 inches. Therefore, in Table 5 for the six board cant, 57 samples equal a width of 1.506116

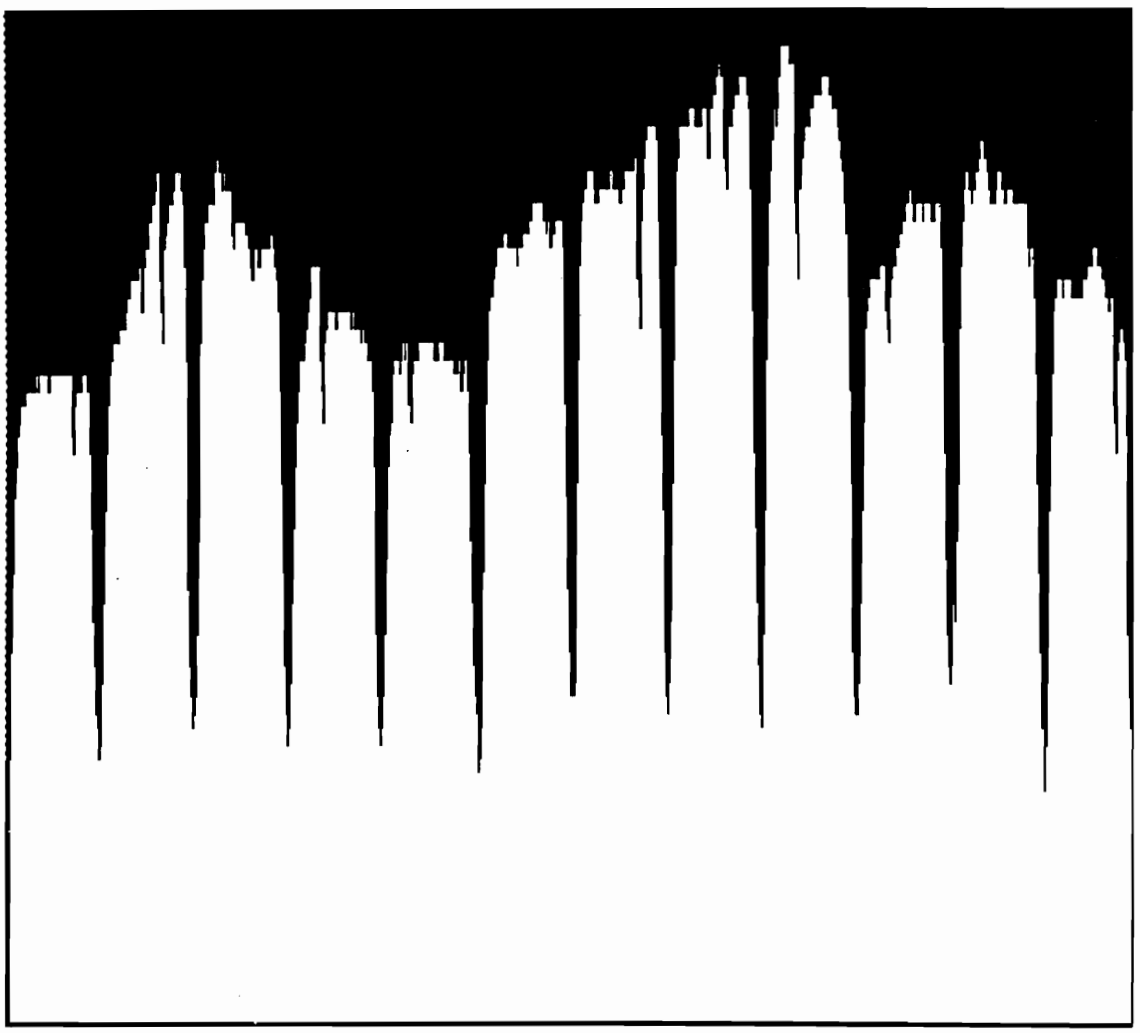


Figure 11. Plot of a reference cant with 12 boards.

Table 7. Results for the change of height.

# OF SAMPLES	COMPUTED VALUE -- inches --	MEASURED VALUE -- inches --
60	1.660635	1.5035
56	1.536091	1.4980
55	1.508414	1.4965
55	1.508414	1.4960
56	1.536091	1.5080
58	1.591146	1.5005
57	1.563769	1.4975
54	1.494575	1.5015
54	1.494575	1.4995
56	1.536091	1.5080
54	1.494575	1.5085
54	1.494575	1.5060

inches. In Table 7, 55 samples equal a width of 1.508414 inches. Both computed values are close to each other, but there is a difference of 2 samples. Even though the single sample width 'sam_len' is the same for both cases, the height change creates the width difference.

From Figure 11, we can see the intensity distribution for each board. In the case of the 12 board cant, intensity smoothing was not done. We can see a huge variation between the computed and measured values. Even though the boards have very similar measured widths, the reflectance variation across the cant fixes a different threshold for each board. In addition, the distribution of later points influences the result. For example, for the first board all the sample points have similar intensity values. Because the initial sample points are low enough to fix a low threshold, most of the points are counted for object presence. However, the sample values for the next board have a much larger variation. This results in a different threshold and different number of sample points being counted for the object presence.

QUALITY CONTROL PROGRAM:

The data from file 'data' are used to run the quality control portion of the system. For this example, the data in the file are assumed to be from two cants, and there are six boards in each cant. The widths are computed in four places along the length of the board. The data are put in a three-dimensional matrix, 'hyper_matrix', where dim1 is the number of cants, dim2 is the number of measurements taken along the length of the board, and dim3 is the number of boards in the cant. These are 2, 4, and 6 in our case. Data in the input file are listed in Table 8. Control limits to check the measurements are entered next. For the data in Table 8, the boards had computed widths between 1.44 and 1.48 inches. Therefore, the upper control limit and lower control limit are initially entered as 1.5 and 1.4.

The list of questions asked to start the execution of the program is given in Table 9. The answers for the example run are given in brackets. When '1' is entered, the system enters into the quality control routines using the data values in the three-dimensional matrix. The data file listed in Table 8 has 8 sets of readings each with six board widths. At least two sets of the

Table 8. Data file for the quality control program.

Data from the input file 'data':

```
1.479459        1.466130        1.532772        1.479459
1.466130        1.372831

1.532772        1.226218        1.452802        1.532772
1.252875        1.426145

1.532772        1.479459        1.479459        1.479459
1.452802        1.399488

1.506116        1.506116        1.479459        1.506116
1.452802        1.452802

1.532772        1.532772        1.466130        1.532772
1.452802        1.426145

0.559795        1.479459        1.506116        1.532772
1.466130        1.426145

1.479459        1.466130        1.532772        1.479459
1.466130        1.372831

1.532772        1.479459        1.479459        1.479459
1.452802        1.399488
```

Table 9. Quality control program initial questions list.

```
Enter # of cants, # of readings & # of boards : ( 2 4 6)

Do you want to change the control limits, Enter

         'y' or 'n' : ( Y )

Enter Control limits : ( 1.5 1.4)

Upper control limit  = 1.5000000

Lower control limit  = 1.4000000


Press key 1 to enter : (1)
```

camera data are required for the quality control program.

The first routine starts when a '2' is entered in response to the next question, and the system goes into the routine 'tf_ave()' and asks about the option to 'BEEP' for out-of-control points. Individual readings are then checked against the control limits, and the results displayed with any out-of-control points highlighted. The example is shown in Table 10. (Out-of -control points are underlined since highlighting can't be shown.) The overall board average and standard deviation for the entire data are computed and displayed. If any other key is pressed, then this whole routine is skipped. Now the system enters into the routine 'board_ave()' to compute the average for each board. The routine asks about the 'BEEP' option, reads the width measurements from 'hyper_matrix', and computes the board averages. During this process, the readings are checked against the limits of 1.4 and 1.5. These limits are different from control limits because they are fixed to eliminate the readings due to knots, wedges and other defects. This is done to stop erroneous readings from entering into the computations and pulling down the results. Readings falling outside these values are ignored. If all the readings of a board fall

Table 10. Results for all quality control check readings.

2 foot averages:

cant #: 1 Reading #: 1

1.479459	1.466130	1.532772	1.479459	1.466130
		1.372831		

cant #: 1 Reading #: 2

1.532772	1.226218	1.452802	1.532772	1.252875
		1.426145		

cant #: 1 Reading #: 3

1.532772	1.479459	1.479459	1.479459	1.452802
		1.399488		

cant #: 1 Reading #: 4

1.506116	1.506116	1.479459	1.506116	1.452802
		1.452802		

cant #: 2 Reading #: 1

1.532772	1.532772	1.466130	1.532772	1.452802
		1.426145		

cant #: 2 Reading #: 2

0.559795	1.479459	1.506116	1.532772	1.466130
		1.426145		

cant #: 2 Reading #: 3

1.479459	1.466130	1.532772	1.479459	1.466130
		1.372831		

cant #: 2 Reading #: 4

1.532772	1.479459	1.479459	1.479459	1.452802
		1.399488		

Board average = 1.464649
All board std.dev = 0.017470

outside these values, then all the readings are used to compute the average of that particular board. The resulting average values are put in a two-dimensional matrix called 'average', and these average values are checked against the given control limits. The results are displayed with or without the 'BEEP' and highlighting. The example results given in Table 11 show all the averages are inside the control limits.

When the selection menu shown in Table 12 pops on the screen, entering option '1' results in board standard deviations being computed by the routine 'board_dev()' and displayed and the system execution ended. Example results are given in Table 13. The variation in the values and magnitude of the values are due to the repeated readings used to demonstrate the system.

If option is '2' is chosen, then new control limits are calculated using the computed averages and standard deviations by executing the routine 'ccl_ck()'. The computed average values of the boards are then checked against these control limits. With or without the 'BEEP' option, the results are displayed with out-of-control points highlighted. Sample results are listed in Table 14.

The routine 'w_cant()' is executed by selecting

Table 11. Computed board averages.

BOARD AVERAGES:

cant # : 1

1.479459 1.472795 1.470573 1.479459 1.457245

1.439473

cant # : 2

1.479459 1.475016 1.472795 1.479459 1.459466

1.426145

Table 12. List of options menu.

SELECT THE OPTION:

1. Board Standard Deviation
2. Check averages by calculated control limits
3. Within cant between boards std.dev
4. Between cant within saw pocket boards std.dev
5. n Board averages

PRESS KEY --> (example press '1' from keyboard for option 1)

Table 13. Board standard deviations.

BOARD STANDARD DEVIATION:

cant # : 1

0.056547 0.009425 0.015390 0.059606 0.007695

0.018849

cant # : 2

0.652482 0.007696 0.009425 0.075396 0.007695

0.059607

Table 14. Results of computed control limits check.

Overall average, X = 1.465945

Standard deviation, S = 0.192065

Upper control limit: 2.042141

Lower control limit: 0.889750

BOARD AVERAGES:

cant # : 1

1.479459 1.472795 1.470573 1.479459 1.457245

1.439473

cant # : 2

1.479459 1.475016 1.472795 1.479459 1.459466

1.426145

---

option '3'. This routine computes the within cant, between board standard deviation for each cant. Example results are listed in Table 15. By selecting option '4', routine 'bet_cant()' is executed. Here between cant, within saw pocket standard deviations are computed and displayed as shown in Table 16. Some data points are shown as zero due to the nature of the demonstration list used.

Option '5' executes the routine 'n_ board()', which gives averages for the last n boards. A series of questions are asked to select the desired results, examples of which are shown in Tables 17, 18, 19, and 20.

POWER MEASUREMENT:

Figure 12 shows a sample plot of the power meter data read by the program 'power.c'. The analog channel resolution has been set for low noise mode, the horizontal screen is divided by 500 pixels, and the maximum number of samples, 'MAX_SIZE', read is fixed at forty. This later variable can be changed to suit the application. The plot is in the form of a control chart to monitor the power consumption. If the number of points increases to 500, the screen will get reset and

Table 15. Within cant between boards standard deviations.

Within cant between boards standard deviation:

cant #	std.dev
	- inches -
1	0.000201
2	0.000353

Table 16. Between cant within saw pocket standard deviations.

Between cant within saw pocket boards std.dev:

Saw pocket #	std.dev
	- inches -
1	0.000000
2	0.001111
3	0.001111
4	0.000000
5	0.001111
6	0.006664

Table 17. n board questions list.

Do you want n board averages? type yes or no: (y)

Do you want last n cant details? type y or n : (y)

How many cants? : (2)

# of cants = 2

cant #: 1

1.479459 1.472795 1.470573 1.479459 1.457245

1.439473

cant #: 2

1.479459 1.475016 1.472795 1.479459 1.459466

1.426145

Table 18. Last cant averages.

How many cants? : (1)

# of cants = 1

Do you want all boards of last cant? type y or n : (y)

1.479459 1.475016 1.472795 1.479459 1.459466

1.426145

Table 19. Last three boards averages.

Do you want all boards of last cant? type y or n: (n)

How many boards?: (3)

1.479459 1.459466 1.426145

Table 20. n saw pocket averages.

Do you want n board averages? type yes or no : (y)

Do you want last n cant details? type y or n : (n)

Last n saw pocket details? type y or n: (y)

How many saw pockets? : (4)

# of saw pockets = 4

pocket # : 3

1.470573 1.472795

pocket # : 4

1.479459 1.479459

pocket # : 5

1.457245 1.459466

pocket # : 6

1.439473 1.426145

last one pocket averages:

1.439473 1.426145

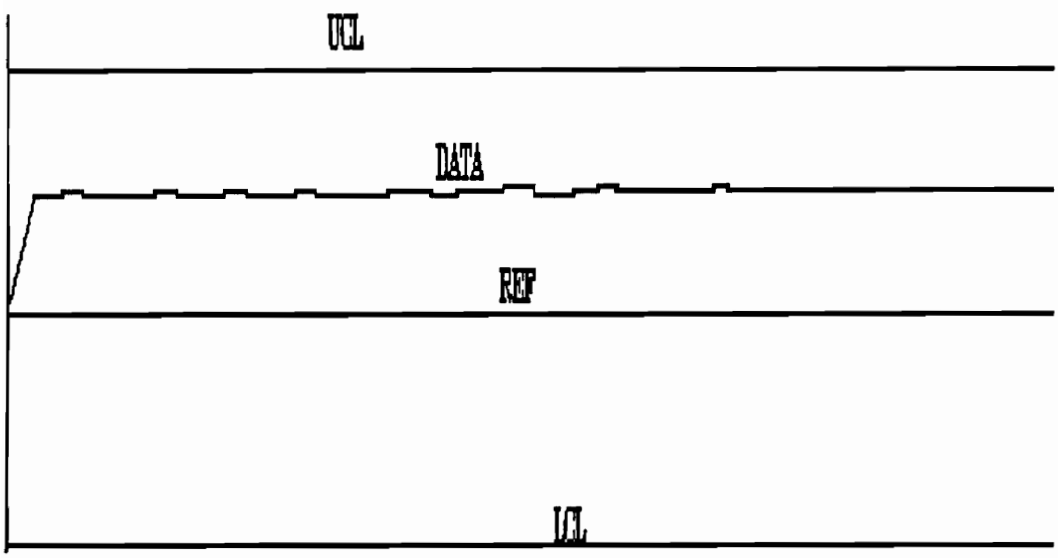


Figure 12. Plot of power consumption data.

the plotting will continue. The channel range is set for '4', but this has to be changed for each range of application. Control limits are entered during the start of the execution of the program. After entering the control limits, the option is to plot the data or tabulate the data. After the data has been tabulated, it is checked against the control limits, and displayed with or without the 'BEEP' option and with out-of-control points highlighted. The load resistor used at the output of the power meter is one kilo-ohms, but the actual value measured is 993 ohms. Therefore, the actual value computed for plotting or tabulation is,

$$\text{analog}[0] = \frac{600 * \text{analog}[0]}{.993 * 1.4142}$$

where analog[0] in the numerator is the output from the A/D card.

CONCLUSIONS

A software technique was developed in this research that can help sawmill personnel monitor the status of their mill production process on a real time basis. In this technique, the widths of the boards sawn by the production system are measured along their lengths in different places and their averages are computed. These measured averages are then checked against quality control limits. By studing the results, the status of the production process can be determined and maintenence decisions made accordingly.

The accuracy achieved in measuring board widths in a static situation was 15 thousandths of an inch. The accuracy may be improved by making finer adjustments to the camera position and better arrangements of the light distribution over the surface of the object. The error introduced by the oscilloscope can not be avoided, but would be eliminated by having a single interface card for the camera to the computer. If that were the case, the image would be processed on a pixel basis rather than the sample basis used in the present case.

The programs for the individual steps involved in this technique are only tested for the present lab

setup. The present setup does not facilitate the continuous running of the system. Once the proper hardware to continuously acquire the data has been acquired, integrating all the programs would result in an on-line system. If the system is moved to a mill, the camera needs to be properly positioned and the required parameters in the formulas used in the programs must be properly adjusted as discussed in the previous sections.

The program developed to monitor the on-line power consumption of the saw has been sucessfully tested for the present lab setup but can be used in a real industrial environment. The two important factors to be changed are the serial load resistor to the output of the meter and the range must be fixed to suit the power range in use.

BIBLIOGRAPHY

1. Brown,T.D. 1986. Lumber size control. Special publication 14, Forest Research Lab, Oregon State University, Corvallis, Oregon. July.

2. Ho,C.S. 1983. Precision of digital vision system. IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-5 (6). Nov.

3. Juran,J.M. 1974. Quality control hand book. Third Ed, Mcgraw Hill Publication, New York.

4. Lee,J.S. 1983. Edge detection by partitioning. Statistical Image Processing and Graphics Edited by Edward J. Wegman and Douglas J. DePriesl, Vol. 72. Naval Research Laboratory, Washington, DC.

5. Li,Y.S., T.Y. Young, and J.A. Magerl. 1988. Subpixel edge detection and estimation with a microprocessor-controlled line scan camera. IEEE Transactions on Industrial Electronics Vol. 35 (1). Feb.

6. Li,Y.S., T.Y. Young, and C.C.Huang. 1989. Noncontact measurement using line scan cameras: analysis of positioning error. IEEE Transactions on Industrial Electronics, Vol. 36 (4). Nov.

7. Mastin,G.A. 1985. Adaptive filters for digital image noise smoothing: an evaluation. Computer Vision, Graphics and Image Processing, Vol. 31, pp.103-121.

8. Microsoft Corporation. 1987. Microsoft Paintbrush: User's Guide. pp.83-90. Redmond, WA.

9. Microsoft Corporation. 1988. C For Yourself. p.102. Redmond, WA.

10. Panda,D.P. 1978. Statistical properties of threshold images. Graphical and Binary Image Processing and Applications by J. C. Stoffel, pp.139-157. Artech House Inc., Dedham, MA.

11. Paumi,J.D. 1988. Laser vs camera inspection in the paper industry. Tappi Journal, Vol.71 pp.129-135. Nov.

12. Rafael,C.G. and Wintz,P. 1987. Image segmentation. Digital image processing (IInd Ed) pp.331-389. Addison- Wesley Publishing Company, Menlo Park, CA.

13. Sternberg,S.R. 1986. Grayscale morphology. Computer Vision, Graphics and Image Processing, Vol. 35 pp.333-355.

14. Weszka,J.S. 1978. A survey of threshold selection techniques. Graphical and Binary Image Processing and Applications by J. C. Stoffel, pp.259-261. Artech House Inc, Dedham, MA.

APPENDICES

APPENDIX A

LIST OF VARIABLES

VARIABLES DESCRIPTION

PROGRAM "log.c":

data[]	: Data sample values in ASCII.
n	: Number of sample points.

PROGRAM "plot.c":

MAX	: Number of sample points to be plotted.
n	: Number of data points in the file.

PROGRAM "width.c":

MAX	: Maximum # of samples in input data.
WL	: Window length for board filter.
WN	: # of boards expected in a run.
SW	: Window length for sigma filter.
ave & std_dev	: mean & std dev for sigma filter.
ul[],ll[]	: 2 sigma points for sigma filter.
LAP	: Window length for velley point detection.

```
          n          : # of samples in the input file.
          mat[]      : # of samples without background.
          bk_pt[]    : Edge points detected.
        rep_len[]    : Repetation window width for edge
                       detection.
          thres[]    : Threshold level for a board.
          array[]    : # of samples in a board.
         det_err     : Sampling error.
         cor_fac     : Correction factor.
         width[]     : Width of a board.
```

PROGRAM "q_con.c":

```
        MAX_DIM1     : # of cants.
        MAX_DIM2     : # of readings in each board.
        MAX_DIM3     : # of boards in a cant.
  max1, max2, max3   : key board inputs to above three
                       dimensions.
        UCL          : Upper control limit.
        LCL          : Lower control limit.
     val1 & val2     : key board inputs to UCL & LCL.
hyper_matrix[][][]   : Three dimensional matrix for
                       input.
 average[][]         : Matrix for board averages.
      Sc[][]         : Matrix for board standard dev.
     al_mn           : Grand mean for all 2 foot
                       readings.
```

std_dev	: Grand std_dev for all readings.
UL_X & LL_X	: Computed control limits.
Sw[]	: within cant bet.board std dev.
Sb[]	: Within saw pocket between board standard deviation.

PROGRAM "power.c":

ANACNT	: # of analog channels.
DIGCNT	: # of digital channels.
MAX_SIZE	: Maximum data points.
UCL & LCL	: Control limits.
val1 & val2	: Key board input to cls.
val3	: key board input to ref. level.
referrence	: Ref level for control chart.
digital[]	: Digital channel array.
analog[]	: Analog channel array.

APPENDIX B

PROGRAM LISTINGS

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                               */
/*  log.c: Acquire single sweep data and log it in               */
/*         a file as ASCII values.                               */
/*                                                               */
/*        Developed the programme following <PC-488>             */
/*        manual.                                                */
/*                                                               */
/*                                                               */
/*        AUTHORS: Milo Clauson                                  */
/*                 Fatima Aslam                                  */
/*                                                               */
/*        Version: V1.0.       903101                           */
/*                                                               */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include <stdio.h>
#include <ms-c488.h>


void err_enter();
void err_send();
void err_xmt();
unsigned int findgpib();



int status;       /* error status returned by f()    */

unsigned int CECaddr;       /* CEC card address      */
int level = 0;             /* GPIB bus level setting */
int addr = 21;          /* GPIB address for IBM card */
int add_tek  = 17;      /* GPIB address of TEK2430   */


#define bsize 7000



char data[bsize];

FILE *fp1;

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

void main(argc, argv) int argc; char *argv[]; {
   int i, n;
```

```
                    /* Find the GPIB card    */

if ((CECaddr = findgpib()) == 0) {
     printf("CEC card not found.\n");
     exit(0);
     }

if (pc488_se(CECaddr)) {
     printf("PC488SE(0x%04X) failed.\n", CECaddr);
     exit(0);
     }
initiali(&level, &addr); /* Initialize GPIB card  */


if (argc < 2) {
  printf ( "\n\nEnter 'log <file name> to run.");
  exit(0);

}
if ((fp1 = fopen(argv[1], "w")) == NULL) {
     printf("Can't open the data file for write.");
     exit(0);
             }

/* Send 8 bit data value for 1024 data points. */

send(&status, "curve?", &add_tek);

     memset(data, ' ', bsize);

     data[bsize-1] = 0;
enter(&status, &add_tek, &n, data);
     if (status != 0)
     err_enter(status);
     else if (n) {
               while(data[i] != 0 && i < n) {
          if (data[i] == ',')
               fputc(' ', fp1);
                    else
                    {
                        fputc(data[i],fp1);
                    }
                        i++;
                 }
                 fputc('\n', fp1);

                }

          fclose(fp1);
```

```
    }

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
        /*          errors for transmit()          */


char *te[] = {
    "(bit 0) Illegal syntax.",
    "(bit 1) Tried to send but not talker.",
    "(bit 2) A quoted string of END in the data string.",

    "(bit 3) Bus timeout or no acceptors.",
    "(bit 4) Bad command mnemonic.",
    "(bit 5) Unknown error code.",
    "(bit 6) Unknown error code.",
    "(bit 7) Unknown error code."
    };

void err_xmt(k) int k;{
    register int j;
    register unsigned int i;
    char ss[10];

    i = (unsigned) k;
    if (i & 0x8F) strcpy(ss, "WARNING:");
    else        strcpy(ss, "ERROR:");

    for (j = 0; j < 8; j++) {
         if ( i & 0x01)
              printf("TRANSMIT: %s %s \n", ss, te[j]);
         i = i >> 1;
         }
    }

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
      /* enter() status error to parse */

void err_enter(i) int i; {
    char ss[64];

    switch(i) {
         case 0: return;
         case 8: sprintf(ss, "Bus time out"); break;
         default: sprintf(ss, "Unknown error # %d", i);
         }
    fprintf(stdout, "ERR ENTER: %s \n", ss);
    }


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```

```
    /* send() status error to parse */

void err_send(i) int i; {
   char ss[64];

   switch(i) {
        case 0: return;
        case 8: sprintf(ss, "Bus time out"); break;
        default: sprintf(ss, "Unknown error # %d", i);
        }
   fprintf(stdout, "ERR SEND: %s \n", ss);
   }

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
            /*   Returns CEC card address     */

unsigned int findgpib() {   /* Find the GPIB card    */
   unsigned int loc;

   for (loc = 0x4000; loc < 0xF000; loc  += 0x400) {
        if ( (peek(loc ,50) == 'C') &&
             (peek(loc ,51) == 'E') &&
             (peek(loc ,52) == 'C') ) {
             return(loc);
             }
        }
   fprintf (stdout,"\nNo PC-488 board installed.\n");
   return(0);
   }

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                               */
/* plot.c: Reads the data from the data file and plots           */
/*         the data using Hercules graphics.                     */
/*                                                               */
/*   Author: Fatima Aslam                                        */
/*   Version: V1.0     900201                                    */
/*                                                               */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include <stdio.h>
#include <stdlib.h>
#include <graph.h>
#include <math.h>
#include <conio.h>
#include <dos.h>
#include <ctype.h>


void graphics_mode(void);
void end_program(void);


struct videoconfig myscreen;
int maxx;
float maxy;
int newx( int );
float newy( float );

#define MAX 1024

main( int argc, char *argv[] )  {

            int i;
            FILE *fp1;
            float n;


        if ( argc < 2 ) {
            printf ( "\n\n Enter 'plot < data filename >'
                        to run:\n\n" );
            exit(0);
        }


        graphics_mode();
```

```
        _setvieworg( 0, newy( 70.0 ));
        _setlinestyle( 0x5555 );
        _moveto( 0, 0 );
        _lineto ( 0, newy( 30.0 ));
        _moveto( 0, 0);
        _lineto ( 0, newy( -70.0 ));
        _moveto( 0, 0);
        _lineto (newx(1050), 0);
        _moveto( 0, 0);



        _setlinestyle( 0xFFFF);
        if ((fp1 = fopen( argv[1], "r")) != NULL)
        {
                for ( i = 0; i < MAX; i++ ){
                   fscanf ( fp1, "%10f", &n);
                   _moveto(newx(i), newy(-n) );
                   _lineto(newx(i), newy(0) );
                }
        }
        else
        {
                printf("Can't open the data file to
                        read.");
                exit(0);
        }



           fclose(fp1);

           end_program();
          _displaycursor( _GCURSORON);



}


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
          /* set graphic mode */

void graphics_mode(void)
{
    _getvideoconfig ( &myscreen );
    _setvideomode ( _HERCMONO );
    _getvideoconfig( &myscreen );
    maxx = myscreen.numxpixels - 1;
    maxy = myscreen.numypixels - 1;
```

```
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
            /* set  # of pixel for x - axis */

int newx( int xcoord )
{
    int nx;
    float tempx;
    tempx = ((float)maxx)/1050;
    tempx = ((float)xcoord) * tempx + 0.5;
    return( (int)tempx );
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
        /* set # of pixels for y _ axis */

float newy ( float ycoord )
{
    int ny;
    float tempy;
    tempy = ((float)maxy)/ 100.0;
    tempy = ((float)ycoord) * tempy + 0.5;
    return( (float)tempy );
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
          /* end graphic mode */

void end_program( void )
{
     getch();
     _setvideomode( _DEFAULTMODE );
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                               */
/*  fplot.c: Reads the data from the data file and               */
/*           plots the data using Hercules Graphics.             */
/*                                                               */
/*                                                               */
/*   Author: Fatima Aslam                                        */
/*                                                               */
/*   Version: V1.0     900201                                    */
/*                                                               */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */


#include <stdio.h>
#include <stdlib.h>
#include <graph.h>
#include <math.h>
#include <conio.h>
#include <dos.h>
#include <ctype.h>


void graphics_mode(void);
void end_program(void);


struct videoconfig myscreen;
int maxx;
float maxy;
int newx( int );
float newy( float );

#define MAX 1024

main( int argc, char *argv[] )  {

            int i, j;
            FILE *fp1;
            float n;

                    j = 0;

 if ( argc < 2 ) {
   printf ( "\n\n Enter 'fplot < data filename >'
             to run:\n\n" );
            exit(0);
  }
```

```
        graphics_mode();
       _setvieworg( 0, newy( 70.0 ));
       _setlinestyle( 0x5555 );
       _moveto( 0, 0 );
       _lineto ( 0, newy( 30.0 ));
       _moveto( 0, 0);
       _lineto ( 0, newy( -70.0 ));
       _moveto( 0, 0);
       _lineto (newx(500), 0);
       _moveto( 0, 0);


      _setlinestyle( 0xFFFF);
    if ((fp1 = fopen( argv[1], "r")) != NULL)
    {
         for ( i = 0; i < MAX; i++ ){
                   fscanf ( fp1, "%10f", &n);

              if ( n > 12.00 ) {
                   _moveto(newx(j), newy(-n) );
                   _lineto(newx(j), newy(0) );
                      j++;
               }
          }
     }
     else
     {
         printf("Can't open the data file to read.");
         exit(0);
     }



          fclose(fp1);

          end_program();
         _displaycursor( _GCURSORON);


}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

void graphics_mode(void)
{
    _getvideoconfig ( &myscreen );
```

```
    _setvideomode ( _HERCMONO );
    _getvideoconfig( &myscreen );
    maxx = myscreen.numxpixels - 1;
    maxy = myscreen.numypixels - 1;
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

int newx( int xcoord )
{
    int nx;
    float tempx;
    tempx = ((float)maxx)/500;
    tempx = ((float)xcoord) * tempx + 0.5;
    return( (int)tempx );
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */


float newy ( float ycoord )
{
    int ny;
    float tempy;
    tempy = ((float)maxy)/ 100.0;
    tempy = ((float)ycoord) * tempy + 0.5;
    return( (float)tempy );
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

void end_program( void )
{
     getch();
     _setvideomode( _DEFAULTMODE );
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                               */
/*  width.c: Reads camera data from the given file,              */
/*           detects the object and computes the                 */
/*           width of the boards. Logs data in a                 */
/*           given file.                                         */
/*                                                               */
/*                                                               */
/*    AUTHOR  : Fatima Aslam                                     */
/*    Version : V1.0                    900902                   */
/*                                                               */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <math.h>
#include <conio.h>
#include <graph.h>


#define MAX 1024
#define WL 70
#define WN 15
#define SW 7
#define LAP 10


main( int argc, char *argv[])
{

            int i, j, k, l, m, num;
            FILE *fp1, *fp2;
            float n,  sam_len;
            int array[WN];
            float width[WN];
            float sf;
            float ul[WN];
            float ll[WN];
            int samples, len;
            float mat[MAX], mat_dif[MAX];
            float ave, dif, std_dev, cor_fac;
            int bk_pt[WN], rep_len[WN], row,  repeat;
            float thres_ave, thres[WN];
            int window, shift, st, col;
```

```
        float det_err;

       memset(mat, 0, MAX * sizeof(float));
       memset(mat_dif, 0, MAX * sizeof(float));
       memset(width, 0, WN * sizeof(float));
       memset(array, 0, WN * sizeof(int));
       memset(bk_pt, 0, WN * sizeof(int));
       memset(rep_len, 0, WN * sizeof(int));
       memset(ul, 0, WN * sizeof(float));
       memset(ll, 0, WN * sizeof(float));
       memset(thres, 0, WN * sizeof(float));


           j = 0;
           k = 0;
           m = 0;
           len = 0;  det_err = 0.0;
           num = 0;
           ave = 0.0; cor_fac = 0.0;
           dif = 0.0;
           std_dev = 0.0; sf = 0.0;
           st = 0;
           repeat = 0; window = 0;
           thres_ave = 0.0; shift = 0;


            /*     Open file to read      */


       _clearscreen(_GCLEARSCREEN);
   if ( argc < 3 ) {
 printf ( "\n\n Enter 'width < data filename> <
         filename >' to run:");

       exit(0);
   }


     printf ( "\n Enter Camera distance to the
               object: ");

     scanf ( "%f", &sf );

if ((fp1 = fopen( argv[1], "r")) != NULL){

  for ( i = 0; i < MAX; i++ ){
               fscanf ( fp1, "%10f", &n);

      /*    filter background data.      */
```

```
            if ( n > 12.0 ){

                        mat[m] = n;
                        m++;
             }
       }
    }
    else
    {
       printf("\nCan't open the data file to read.\n");
             exit(0);
     }

            fclose(fp1);
             samples = m ;


#undef MAX
#define MAX samples


             /*  take away the ref board data */

    for ( m = 0; m < MAX; m++ ) {
          mat [len] = mat[m];
                len++;
    }
             printf ( "\n%d\n", len );
             num = len/(WL-5);
                     num = (int)num + 1;
             printf ( "\n%d\n", num );


#undef WN
#define WN num


       /*  detect real boards & computes average &
              cls to sigma filter. */

                    rep_len[0] = SW;
  for ( row = 0; row < WN; row++ ) {
      for ( len = bk_pt[row]; len < rep_len[row];len++ ) {
              ave += mat[len];
      }

          ave = ave/(rep_len[row] - bk_pt[row]);
        if ( ave > 14.000 ){
              repeat++;
        }
```

```
   for ( len = bk_pt[row]; len < rep_len[row];len++) {
            mat_dif[len] = mat[len] - ave;
            dif += (mat_dif[len] * mat_dif[len]);
   }


            std_dev = sqrt( dif/ ((rep_len[row]
                          - bk_pt[row]) - 1));

            ul[row] = ave + (2 * std_dev);
            ll[row] = ave - (2 * std_dev);

      /* detects edge for each board  */

             shift = bk_pt[row] +  WL;
                    st = 0;
  for ( len = (shift - LAP); len < shift; len++) {
    if( mat[len -1] <= mat[len]) {
        printf ( "%10f", mat[len - 1] );
        bk_pt[row + 1] = len + 1;
        break;
      }
      else
      {
          if ( mat[len-1] > mat[len] ) {
                   st++;
          }
           if ( st > 8 ) {
              printf ( "%10f", mat[len - 7] );
                bk_pt[row + 1] = (len - 7);
                          break;
           }

     }
 }

      col = 0;
for ( len = bk_pt[row + 1];
           len < (bk_pt[row+1]  + 10); len++) {

  if(( mat[len - 1] <= mat[len]) &&
               ( mat[len - 1] < 20.0 ) ){

         col++;
     if ( mat[len - 1] == 0.0) {
          break;
     }
  }
}
```

```
     if ( col > 3 ) {
          bk_pt[row + 1] = bk_pt[row + 1] + col - 1;
     }
          printf ( "%5d\n", bk_pt[row + 1] );
          rep_len[row + 1] = bk_pt[row + 1] + SW;
          ave = 0.0; dif = 0.0;

 }


#undef WN
#define WN repeat


            /*   find the threshold for each board. */

 for ( row = 0; row < WN; row++ ) {
            for ( len = bk_pt[row];
                    len < rep_len[row]; len++) {

                if ( (mat[len]  > ll[row]) &&
                        ( mat[len] < ul[row])) {

                     thres_ave += mat[len];
                       window++;
                 }
             }

             thres_ave = thres_ave/window;
             printf ( "%10f", thres_ave );

    if ( (thres_ave > 25 ) && ( thres_ave <= 50 )) {
      if ( (thres_ave > 25 ) && (thres_ave < 35)) {
         thres_ave = thres_ave -3.00;
      }
      else
         thres_ave = thres_ave - 5.00;
    }
    else
    {
     if ( (thres_ave > 50 ) && ( thres_ave <= 60) ) {
          thres_ave = thres_ave - 10.00;
     }
     else
     {
          if ( thres_ave > 60 )
          thres_ave = thres_ave - 20.00;
     }
    }
```

```
          thres[row] = thres_ave;
          printf ( "%10f\n", thres[row] );
               window = 0; thres_ave = 0;

}

          /* detects # of boards    */

for ( row = 0; row < WN; row++ ) {
     for ( len = bk_pt[row];
               len < bk_pt[row + 1]; len++) {

               if ( mat[len]  >= thres[row] )  {
                    j++;
                 }
      }
               if ( j > 0)    {
                    array[k] = j;
                    k++;
                }

                      j = 0;

}

          printf ( "No of boards detected, k = %d\n", k);

                        l = k;


#undef WN
#define WN l

       /* compute width, display nad log the data */

        sam_len = ( 24 * 25 * 20.48)/( 42 * 1024 * 16.8);

          printf ( "\n%10f\n", sam_len );

               det_err = (sf * sam_len)/(2 * 25);

printf ( "\n\n       Board #       Samples    width \n");

printf ( "     -------       -------        -----\n\n" );


 if ((fp2 = fopen( argv[2], "a")) != NULL){
    for ( k = 0; k < WN; k++) {
       if ( array[k] > 60 )
            printf ( " Board %d is wider\n", (k+1) );
```

```
        else
        {
            if ( array[k] < 50 ) {
printf ( " Board %d is narrow or knot or wider wane.\n",
            (k+1));
        }
     }


     width[k] = ( sf * sam_len * array[k])/ 25;


     if ( (array[k] > 55) && ( array[k] < 60 ) ) {
        cor_fac = (array[k] - .5) * 0.000466435;
        width[k] = width[k] - det_err + cor_fac;
  printf ( "%10d  %15d   %20f\n",(k+1), array[k],
                 width[k] );
        fprintf ( fp2,  "%15f", width[k] );

     }
     else
     {
       if ( (array[k] > 50) && (array[k] < 55 ) ) {
          cor_fac = (array[k] + .5) * 0.000466435;
          width[k] = width[k] + det_err + cor_fac;
    printf ( "%10d  %15d   %20f\n",(k+1), array[k],
           width[k] );
               fprintf ( fp2,  "%15f", width[k] );
        }
        else
        {
          cor_fac = array[k]  * 0.000466435;
     width[k] = width[k] + ((2/3) * det_err) + cor_fac;
     printf ( "%10d  %15d   %20f\n",(k+1), array[k],
        width[k] );
             fprintf ( fp2,  "%15f", width[k] );
         }
     }
   }

  }
  else
  {
     printf("\nCan't open the data file for write.\n");
     exit(0);
   }

           fputc ( '\n', fp2 );
           fclose(fp2);
```

```
}
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                               */
/*  width.c: Reads camera data from the given file,              */
/*           detects the object and computes the                 */
/*           width of the boards. Logs data in a                 */
/*           given file.  The distance from the                  */
/*           object is 40 inches and no threshold                */
/*           smoothing.                                          */
/*                                                               */
/*    AUTHOR  : Fatima Aslam                                     */
/*    Version : V1.0                    900902                   */
/*                                                               */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */


#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <math.h>
#include <conio.h>
#include <graph.h>


#define MAX 1024
#define WL 70
#define WN 15
#define SW 7
#define LAP 10


main( int argc, char *argv[])
{


        int i, j, k, l, m, num;
            FILE *fp1, *fp2;
            float n,  sam_len;
            int array[WN];
            float width[WN];
            float sf;
            float ul[WN];
            float ll[WN];
            int samples, len;
            float mat[MAX], mat_dif[MAX];
            float ave, dif, std_dev, cor_fac;
```

```
        int bk_pt[WN], rep_len[WN], row,  repeat;
        float thres_ave, thres[WN];
        int window, shift, st, col;
        float det_err;

      memset(mat, 0, MAX * sizeof(float));
      memset(mat_dif, 0, MAX * sizeof(float));
      memset(width, 0, WN * sizeof(float));
      memset(array, 0, WN * sizeof(int));
      memset(bk_pt, 0, WN * sizeof(int));
      memset(rep_len, 0, WN * sizeof(int));
      memset(ul, 0, WN * sizeof(float));
      memset(ll, 0, WN * sizeof(float));
      memset(thres, 0, WN * sizeof(float));


          j = 0;
          k = 0;
          m = 0;
          len = 0;  det_err = 0.0;
          num = 0;
          ave = 0.0; cor_fac = 0.0;
          dif = 0.0;
          std_dev = 0.0; sf = 0.0;
          st = 0;
          repeat = 0; window = 0;
          thres_ave = 0.0; shift = 0;


           /*     Open file to read      */


        _clearscreen(_GCLEARSCREEN);
     if ( argc < 3 ) {
printf ( "\n\n Enter 'width < data filename> <
          filename >' to run:");

        exit(0);
     }


printf ( "\n Enter Camera distance to the object: ");
     scanf ( "%f", &sf );

if ((fp1 = fopen( argv[1], "r")) != NULL){

    for ( i = 0; i < MAX; i++ ){
          fscanf ( fp1, "%10f", &n);

       /*    filter background data.      */
```

```
            if ( n > 12.0 ){

                  mat[m] = n;
                  m++;
            }
         }
      }
      else
      {
         printf("\nCan't open the data file to read.\n");
          exit(0);
      }

          fclose(fp1);
          samples = m ;


#undef MAX
#define MAX samples


              /*  take away the ref board data */

          for ( m = 0; m < MAX; m++ ) {
            mat [len] = mat[m];
                    len++;
          }
               printf ( "\n%d\n", len );
               num = len/(WL-5);
               num = (int)num + 1;
               printf ( "\n%d\n", num );


#undef WN
#define WN num


         /*  detect real boards & computes average &
             cls to sigma filter. */

                       rep_len[0] = SW;
  for ( row = 0; row < WN; row++ ) {
    for ( len = bk_pt[row]; len < rep_len[row]; len++ ) {
                ave += mat[len];

    }
          ave = ave/(rep_len[row] - bk_pt[row]);
          if ( ave > 16.000 ){
                 repeat++;
```

```
        }

    for ( len = bk_pt[row]; len < rep_len[row]; len++) {
            mat_dif[len] = mat[len] - ave;
            dif += (mat_dif[len] * mat_dif[len]);
    }


std_dev = sqrt( dif/ ((rep_len[row]  - bk_pt[row]) - 1));
            ul[row] = ave + (2 * std_dev);
            ll[row] = ave - (2 * std_dev);

        /* detects edge for each board  */

            shift = bk_pt[row] +  WL;
                  st = 0;
      for ( len = (shift - LAP); len < shift; len++) {
         if( mat[len -1] <= mat[len]) {
            printf ( "%10f", mat[len - 1] );
            bk_pt[row + 1] = len + 1;
            break;
         }
         else
         {
           if ( mat[len-1] > mat[len] ) {
                       st++;
         }
              if ( st > 8 ) {
                printf ( "%10f", mat[len - 7] );
                bk_pt[row + 1] = len - 7;
                       break;
              }

      }
   }

        col = 0;
   for ( len = bk_pt[row + 1]; len < (bk_pt[row+1] + 10);
                  len++) {
     if(( mat[len - 1] <= mat[len]) && ( mat[len - 1] <
                  20.0 ) ){
            col++;
        if ( mat[len-1] == 0.0 )  {
             break;
        }
     }
   }

     if ( col > 3 ) {
         bk_pt[row + 1] = bk_pt[row + 1] + col - 1;
```

```
    }
        printf ( "%5d\n", bk_pt[row + 1] );
        rep_len[row + 1] = bk_pt[row + 1] + SW;
        ave = 0.0; dif = 0.0;

 }


#undef WN
#define WN repeat


             /*   find the threshold for each board. */

for ( row = 0; row < WN; row++ ) {
  for ( len = bk_pt[row]; len < rep_len[row]; len++) {
   if ( (mat[len]  > ll[row]) && ( mat[len] < ul[row])) {
              thres_ave += mat[len];
               window++;
   }
  }


         thres_ave = thres_ave/window;
         printf ( "%10f", thres_ave );
         thres[row] = thres_ave;
         printf ( "%10f\n", thres[row] );
         window = 0; thres_ave = 0;

}

         /* detects # of boards   */

 for ( row = 0; row < WN; row++ ) {
  for ( len = bk_pt[row]; len < bk_pt[row + 1]; len++) {
        if ( mat[len]  >= thres[row] )  {
            j++;
        }
  }
            if ( j > 0)   {
                 array[k] = j;
                 k++;
            }

                   j = 0;

 }

      printf ( "No of boards detected, k = %d\n", k);
```

```
                    l = k;

#undef WN
#define WN 1

       /* compute width, display nad log the data */

        sam_len = ( 24 * 25 * 20.48)/( 42 * 1024 * 16.8);

          printf ( "\n%10f\n", sam_len );

          det_err = (sf * sam_len)/(2 * 25);

 printf ( "\n\n       Board #      Samples  Width \n" );
 printf ( "       -------      -------    ----- \n\n" );


    if ((fp2 = fopen( argv[2], "a")) != NULL){
       for ( k = 0; k < WN; k++) {
         if ( array[k] > 60 )
             printf ( " Board %d is wider\n", (k+1) );
         else
         {
             if ( array[k] < 50 ) {
printf ( " Board %d is narrow or knot or wider wane.\n",
                 (k+1));
         }
       }


     width[k] = ( sf * sam_len * array[k])/ 25;


     if ( (array[k] > 54) && ( array[k] < 60 ) ) {
        cor_fac = (array[k] - .5) * 0.000466435;
        width[k] = width[k] - det_err + cor_fac;
        printf ( "%10d  %15d   %20f\n",(k+1), array[k],
                    width[k] );
        fprintf ( fp2,  "%15f", width[k] );

     }
     else
     {
       if ( (array[k] > 50) && (array[k] < 54 ) ) {
          cor_fac = (array[k] + .5) * 0.000466435;
          width[k] = width[k] + det_err + cor_fac;
          printf ( "%10d  %15d   %20f\n",(k+1), array[k],
                 width[k] );
```

```
            fprintf ( fp2,  "%15f", width[k] );
          }
          else
          {
            cor_fac = array[k]  * 0.000466435;
        width[k] = width[k] + ((2/3) * det_err) + cor_fac;
            printf ( "%10d  %15d   %20f\n",(k+1), array[k],
                     width[k] );
            fprintf ( fp2,  "%15f", width[k] );
          }
        }
    }

 }
 else
 {
      printf("\nCan't open the data file for write.\n");
           exit(0);
 }

            fputc ( '\n', fp2 );
            fclose(fp2);

}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                               */
/*  bis.c: Contains functions required for q_con.c               */
/*         for screen adjustment.                                */
/*                                                               */
/*   AUTHOR  : Fatima Aslam                                      */
/*   VERSION : V1.0                    892010                    */
/*                                                               */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */


#include <stdio.h>
#include <dos.h>
#include <graph.h>


#define ESC '\x1B'

void printa ( char *string, int attr , int row, int col )

{
   union REGS  cur_regs, write_regs;

        while ( *string )
   {
        cur_regs.h.ah = 2; /* using BIOS set cursor
                              function */
        cur_regs.h.dh = row;
        cur_regs.h.bh = 0;
        cur_regs.h.dl = col++;
        int86 (0x10, &cur_regs, &cur_regs);

        write_regs.h.ah = 9;  /* using BIOS write char
                                 and attr fn */
        write_regs.h.bh = 0;
        write_regs.x.cx = 1;
        write_regs.h.bl = attr;
        write_regs.h.al = *string++;
        int86 ( 0x10, &write_regs, &write_regs);
   }
}
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

void CURSOR (int row, int col)
{
   union REGS reg;
```

```
   reg.h.ah = 2;
   reg.h.dh = row;
   reg.h.dl = col;
   reg.h.bh = 0;
   int86 ( 0x10, &reg, &reg);
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

pos()   /* Return text cursor column position */
{

   union REGS r, *inregs, *outregs;

   inregs = &r;
   outregs = &r;

   r.h.ah = 3;  /* "Read cursor position" video call(3) */

   r.h.bh = 0;  /* page number */

   int86(0x10, inregs, outregs);  /* get position via
                                     interrupt 10h */

   return ((int) r.h.dl);     /* Return column
                                 position in dl */
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

row()    /* Return text cursor row position */
{

   union REGS r, *inregs, *outregs;

   inregs = &r;
   outregs = &r;

   r.h.ah = 3;
   r.h.bh = 0;

   int86(0x10, inregs, outregs);

   return ((int) r.h.dh );     /* return row position */
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

check()
{
```

```
            char sw;

        if ( row() == 23 )
        {
           printa ( "PRESS ESC TO CONTINUE ->",10,24,50);
                 sw = getch();
                 if ( sw == ESC)
                 {
                     _clearscreen(_GCLEARSCREEN);
                 }
        }
}
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                               */
/* q_con.c: Reads width of the boards from a file                */
/*          and runs Quality Control checks.                     */
/*                                                               */
/*    AUTHOR  : Fatima Aslam                                     */
/*    VERSION : V1.0                  900902                     */
/*                                                               */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */


#include  <stdio.h>
#include  <string.h>
#include  <ctype.h>
#include  <conio.h>
#include  < math.h>
#include  <stdlib.h>
#include  <graph.h>
#include  <dos.h>
#include  "bis.c"


void printa ( char *string, int attr, int row, int col );

row();
pos();
check();
void tf_ave (void );
void board_ave(void);
void board_dev(void);
void ccl_ck(void);
void w_cant(void);
void bet_cant(void);
void n_board(void);


#define MAX_DIM1 10
#define MAX_DIM2 20
#define MAX_DIM3 15
#define SQUARE(dif) (dif * dif)
#define CR '\x0d'
#define ESC '\x1B'

int max1, max2, max3;
int dim1, dim2, dim3;
float UCL, LCL, val1, val2;
float hyper_matrix [MAX_DIM1] [MAX_DIM2] [MAX_DIM3];
```

```
float average [MAX_DIM1][MAX_DIM3];
float Sc [MAX_DIM1][MAX_DIM3];
char string[128], inkey;


main( int argc, char *argv[])
{
        FILE *fp1;
        float width;
        char sh, sv, fh;


            /*     Open a file to read data    */

  if ( argc < 2 ) {
     printf ( "\n\n Enter 'qc < data filename >' to
               run:\n\n" );
           exit(0);
  }


          /*   Enter # of cants, reading, boards  */

        _clearscreen(_GCLEARSCREEN);
   printf ( "\n\n\n Enter # of cants, # of Reading &
              # of Boards : " );
        scanf ( "%d%d%d", &max1, &max2, &max3 );

        printa ( " PRESS ENTER TO CONTINUE", 10, 24, 50);

          while (( inkey = getch()) != CR);


   if ( ( fp1 = fopen ( argv[1], "r")) != NULL ) {

       /*   initialize number of cants      */
     for ( dim1 = 0; dim1 < max1; dim1++ ) {

      /* initialize number of reading in each board */
          for ( dim2 = 0; dim2 < max2; dim2++ ) {

           /* initilize number of boards in a cant  */
            for ( dim3 = 0; dim3 < max3; dim3++) {
                fscanf ( fp1, "%15f", &width );
                hyper_matrix [dim1][dim2][dim3] = width;
            }
```

```
                printf ( "\n" );
            }
        }
    }
    else
    {
       printf ( "\n\n Can't open file to read. \n\n" );
    }
          fclose(fp1);


          /* Enter the control limits */

          _clearscreen(_GCLEARSCREEN);
printf ( " Do you want to change the control limits,
              Enter 'y' or 'n' :");
          scanf ("%s", &sv);
        if ( (sv  == 'Y') || (sv == 'y')) {
           printf ( "\n\n\n Enter Control limits : " );
           scanf ( "%f%f", &val1, &val2 );
        }

          UCL = val1;
          LCL = val2;
          printf ( "\nUpper control limit = %f\n", UCL );
          printf ( "\nLower control limit = %f\n", LCL );

          printf ( "\n\n\nPress key 1 to enter :");

     fh = getch();
     printf ( "%c", fh);
if (fh == '1')
{
       printa ( " PRESS ENTER TO CONTINUE", 10, 24, 50);
       while (( inkey = getch()) != CR);



          _clearscreen(_GCLEARSCREEN);
      printf ( "\n\nDo you want 2 foot averages to be
                 checked?");
printf ( "\nIf yes PRESS 2 or any key to come out -> ");

      fh = getch();
      printf ( "%c", fh);

             /*   Compute two foot averages  */

               if ( fh == '2') {
                     tf_ave();
```

```
            }

          /*  computing each board average  */

              board_ave();

          /*   Select option for result      */
    printa ( " PRESS ENTER TO CONTINUE", 10, 24, 50);
    while (( inkey = getch()) != CR);

    _clearscreen(_GCLEARSCREEN);
    printf ( "\n\n\n\n");
    printa ( " SELECT THE OPTION:", 10, 4,1);
    printf( "\n\n");
    printf ( "1. Board Standard Deviation ");
    printf ( "\n2. Check averages by calculated
                control limits");
    printf ( " \n3. Within cant between boards
                std.dev");
    printf ( " \n4. Between cant within saw pocket
                boards std.dev");
    printf ( " \n5. n Board averages \n");
    printf ( " \n   Any other key to return    \n");
    printf ( " \n\n PRESS KEY -> ");

    sh = getch();
    printf ( "%c", sh);

switch (sh) {
   case '1': {
             /*  Board std.dev  */

      printa ( " PRESS ENTER TO CONTINUE", 10, 24, 50);
      while (( inkey = getch()) != CR);
      _clearscreen(_GCLEARSCREEN);
      printa ( "BOARD STANDARD DEVIATION: ", 10, 0,0);
                board_dev();
      for ( dim1 = 0; dim1 < max1; dim1++ ) {
        printf( "\n\ncant # :%d ", (dim1+1));
                check();
        for (dim3 = 0; dim3 < max3; dim3++ ) {
            printf ( "%10f", Sc [dim1][dim3]);
                    check();
        }
      }
                break;
    }
```

```
        case '2': {

           /* Overall board average and std.dev */

        printa ( " PRESS ENTER TO CONTINUE", 10, 24, 50);
        while ((inkey = getch()) != CR);
                ccl_ck();
                break;
        }
        case '3': {

          /* computation for within cant statistics */

        printa ( " PRESS ENTER TO CONTINUE", 10, 24, 50);
        while (( inkey = getch()) != CR);
                 w_cant();
                 break;
        }
        case '4':  {
         /* computation for between cant statistics */

        printa ( " PRESS ENTER TO CONTINUE", 10, 24, 50);
        while (( inkey = getch()) != CR);
                 bet_cant();
                 break;
        }
        case '5': {
              /* n board statistics */

        printa ( " PRESS ENTER TO CONTINUE", 10, 24, 50);
        while (( inkey = getch()) != CR);
                n_board();
                break;
        }
    }
  }
}
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

          /*   Compute two foot averages  */

void tf_ave ( void )
{

            char sw;
            int i;
            float  bth, al_mn, std_dev, sum_dif, dif;

            bth = (float)0;
            al_mn = (float)0;
```

```
            sum_dif = (float)0;
            std_dev = (float)0;
            i = 0;

            UCL = val1;
            LCL = val2;

 printf ( "\n\nDo you want 'BEEP'? type 'Y'es or 'N'o
                    ->");
            scanf("%s",  &sw );
            printf ( "%s", sw);
    if ( (sw == 'Y')  || (sw == 'y')) {
         _clearscreen(_GCLEARSCREEN);
         printa ( "2 foot averages:", 10, 0,0);
       for ( dim1 = 0; dim1 < max1; dim1++ )  {
                check();
                printf ( "\n" );
                check();
         for ( dim2 = 0; dim2 < max2; dim2++ ) {
                  check();
    printf ( "\ncant # :%d, reading #:%d ",(dim1+1),
                  (dim2+1));
              check();
              printf ( "\n" );
              check();
           for ( dim3 = 0; dim3 < max3; dim3++ )  {
                  check();
             if (( UCL > hyper_matrix [dim1][dim2][dim3])
               && ( LCL < hyper_matrix [dim1][dim2][dim3]
                          )) {
      printf ( " %8f ", hyper_matrix[dim1][dim2][dim3] );
                    i++;
               bth += hyper_matrix [dim1][dim2][dim3];
              }
              else
              {
sprintf (string, " %8f  ",hyper_matrix[dim1][dim2][dim3]);
                  printf ( "\a");
                  printa (  string, 10, row(), pos() );
              }
           }


        }
              check();
      }
     }
     else
     {
          _clearscreen(_GCLEARSCREEN);
```

```
          printa ( "2 foot averages:", 10, 0,0);
         for ( dim1 = 0; dim1 < max1; dim1++ )  {
              check();
              printf ( "\n" );
              check();
           for ( dim2 = 0; dim2 < max2; dim2++ ) {
              check();
  printf ( "\ncant #: %d Reading # :%d ",(dim1+1),
               (dim2+1));
              check();
              printf ( "\n" );
              check();
            for ( dim3 = 0; dim3 < max3; dim3++ ) {
                  check();
             if (( UCL > hyper_matrix[dim1][dim2][dim3])
               && ( LCL < hyper_matrix[dim1][dim2][dim3]
                        )) {
    printf ( " %8f ", hyper_matrix [dim1][dim2][dim3] );
                       i++;
               bth += hyper_matrix [dim1][dim2][dim3];
              }
              else
              {
sprintf (string, " %8f  ",hyper_matrix[dim1][dim2][dim3]);

               printa (  string, 10, row(), pos() );
              }
            }
          }
               check();
        }
      }
           /* compute average for all boards */

     printa ( "PRESS ENTER TO CONTINUE", 10, 24, 50);
     while((inkey = getch()) != CR);
     _clearscreen(_GCLEARSCREEN);
     printa ( "\n\n All board statistics: ", 10, 2, 1);
        al_mn = bth/(i);
        printf ( " \n\n Board average = %f\n", al_mn);

        /* compute std.dev for all boards */

      for ( dim1 = 0; dim1 < max1; dim1++) {
        for ( dim2 = 0; dim2 < max2; dim2++)  {
          for ( dim3 = 0; dim3 < max3; dim3++ ) {
            if (( UCL > hyper_matrix [dim1][dim2][dim3])
              && ( LCL < hyper_matrix [dim1][dim2][dim3]
                    )) {
         dif = hyper_matrix [dim1][dim2][dim3] - al_mn;
```

```
                sum_dif += SQUARE(dif);
              }
            }
          }
        }
              std_dev = sqrt(sum_dif/( i - 1));
  printf ( "\n All Board std dev =  %f\n\n", std_dev );

}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

           /*   Compute each board average   */

void board_ave(void)
{

                char ss;
                int i;


memset (average, 0, MAX_DIM1 * MAX_DIM3 * sizeof (float));


            i = 0;

            UCL = val1;
            LCL = val2;
printf ( "\n\nDo you want 'BEEP'? type 'Y'es or 'N'o
                -> ");
    scanf("%s",  &ss );
    printf ( "%s", ss);
    printa ( " PRESS ENTER TO CONTINUE", 10, 24, 50);
    while((inkey = getch()) != CR);
    if ( (ss == 'Y')  || (ss == 'y')) {
        if ( max2 > 18 )  {
           printf ( " Longer board " );
        }

        _clearscreen(_GCLEARSCREEN);
        printa ( "BOARD AVERAGES: ", 10, 0, 0);
             dim1 = 0;
        while ( dim1 < max1 )  {
             check();
          printf ( "\n\ncant # :%d ", (dim1+1) );
             check();
             dim3 =0;
           while ( dim3 < max3 )  {
            for ( dim2 = 0; dim2 < max2; dim2++) {
              if ( ( hyper_matrix [dim1][dim2][dim3] >
```

```
                1.4 ) && ( hyper_matrix [dim1][dim2][dim3]
                 < 1.5 )) {
                        i++;
average [dim1][dim3] += hyper_matrix [dim1][dim2][dim3];

              }
            }
            if ( i == 0 ) {
              for ( dim2 = 0; dim2 < max2; dim2++) {
                          i++;
average [dim1][dim3] += hyper_matrix [dim1][dim2][dim3];

              }
            }

                 check();
average [dim1][dim3] = average [dim1][dim3] / (i);
           if (( UCL > average [dim1][dim3]) &&
               (LCL < average [dim1][dim3])) {
                     check();
        printf ( " %8f ", average [dim1][dim3]);
           }
           else
           {
   sprintf (string, " %8f  ", average [dim1][dim3]);
             printf ( "\a");
             printa (  string, 10, row(), pos() );
           }
                dim3++;
                i = 0;
                check();
        }

                dim1++;
     }
   }
   else
   {
      if ( max2 > 18 ) {
        printf ( " Longer board " );
      }
        _clearscreen(_GCLEARSCREEN);
        printa ( "BOARD AVERAGES: ", 10, 0, 0);
             dim1 = 0;
      while ( dim1 < max1 ) {
            check();
            printf ( "\n\ncant # :%d ", (dim1+1) );
            check();
            dim3 =0;
        while ( dim3 < max3 )  {
```

```
            for ( dim2 = 0; dim2 < max2; dim2++)  {
              if ( ( hyper_matrix [dim1][dim2][dim3] > 1.4
                 ) && ( hyper_matrix [dim1][dim2][dim3] <
                        1.5 )) {
                           i++;
 average [dim1][dim3] += hyper_matrix [dim1][dim2][dim3];

              }
            }
            if ( i == 0 ) {
              for ( dim2 = 0; dim2 < max2; dim2++) {
                        i++;
 average [dim1][dim3] += hyper_matrix [dim1][dim2][dim3];
              }
            }

               check();
      average [dim1][dim3] = average [dim1][dim3] / (i);
            if (( UCL > average [dim1][dim3]) &&
               (LCL < average [dim1][dim3]))  {
                     check();
               printf ( " %8f ", average [dim1][dim3]);
            }
            else
            {
       sprintf (string, " %8f  ", average [dim1][dim3]);
              printa (  string, 10, row(), pos() );
            }
                    dim3++;
                    i = 0;
                    check();
          }
                   dim1++;
       }
     }
}
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

            /* computing each board std.dev */

void board_dev(void)
{

       float ave_dif [MAX_DIM1][MAX_DIM2][MAX_DIM3];
       int i;


memset (Sc, 0, MAX_DIM1 * MAX_DIM3 * sizeof (float));
memset (ave_dif, 0, MAX_DIM1 * MAX_DIM2 * MAX_DIM3 *
          sizeof (float));
```

```
for ( dim1 = 0; dim1 < max1; dim1++) {
  for ( dim3 = 0; dim3 < max3; dim3++) {
              i = 0;
              check();
    for ( dim2 = 0; dim2 < max2; dim2++) {
      if ( ( hyper_matrix [dim1][dim2][dim3] > 1.4 )
         && ( hyper_matrix [dim1][dim2][dim3] < 1.5 )) {
         ave_dif [dim1][dim2][dim3] =
            hyper_matrix[dim1][dim2][dim3] -
                  average [dim1][dim3];
        if ( ave_dif[dim1][dim2][dim3] != 0.000000 ) {
                i++;
        }
          ave_dif [dim1][dim2][dim3] =
             ave_dif [dim1][dim2][dim3] *
                   ave_dif [dim1][dim2][dim3] ;
        Sc [dim1][dim3] += ave_dif [dim1][dim2][dim3];
      }
    }

    if ( (i == 0 )  || ( i == 1 ) ) {
      for ( dim2 = 0; dim2 < max2; dim2++) {
             ave_dif [dim1][dim2][dim3] =
                hyper_matrix [dim1][dim2][dim3] -
                       average [dim1][dim3];

        if ( ave_dif[dim1][dim2][dim3] != 0.000000 )  {
               i++;
        }
        ave_dif [dim1][dim2][dim3] =
            ave_dif [dim1][dim2][dim3] *
                ave_dif [dim1][dim2][dim3] ;
        Sc [dim1][dim3] += ave_dif [dim1][dim2][dim3];
      }
    }

      Sc [dim1][dim3] = sqrt(Sc [dim1][dim3]/(i - 1));

   }
  }
 }
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

      /*  Compute LCL & UCL  for all boards & check  */

void ccl_ck(void)
{
```

```
        float X, S, UL_X, LL_X;
        char  su;


            X = 0.0;
            S = 0.0;
            UL_X = 0.0;
            LL_X = 0.0;

      _clearscreen(_GCLEARSCREEN);
          board_dev();
   for ( dim1 = 0; dim1 < max1; dim1++) {
     for ( dim3 = 0; dim3 < max3; dim3++){
          X += average [dim1][dim3];
          S += ( Sc [dim1][dim3] * Sc [dim1][dim3] );
     }
   }

          X = X / ( max1 * max3 );
          S = sqrt( S / (max1 * max3) );
          printf ( "\n\n X = %8f, S = %8f\n ", X,S );


              /* compute control limits  */

          UL_X = X + 3 * S;
          LL_X = X - 3 * S;
  printf ( " \n\n Upper control limit :%8f", UL_X);
  printf ( " \n\n Lower control limit :%8f\n\n", LL_X );


              /* Check X for control limits */


printf ( "\n\nDo you want 'BEEP'? type 'Y'es or 'N'o
                  -> ");
     scanf("%s",  &su );
     printf ( "%s", su);
     printa ( " PRESS ENTER TO CONTINUE", 10, 24, 50);
     if ( (su == 'Y')  || (su == 'y')) {
         _clearscreen(_GCLEARSCREEN);
         printa ( "BOARD AVERAGES:", 10, 0, 0);
       for ( dim1 = 0; dim1 < max1; dim1++) {
            check();
         printf ( "\n\ncant # :%d ", (dim1+1) );
            check();
         for ( dim3 = 0; dim3 < max3; dim3++)  {
            check();
           if (( UL_X > average [dim1][dim3]) &&
               ( LL_X < average [dim1][dim3] )) {
```

```
            printf ( " %8f ", average [dim1][dim3] );
          }
          else
          {
       sprintf (string, " %8f  ", average [dim1][dim3]);
            printf ( "\a");
            printa (  string, 10, row(), pos() );
          }
        }
      }
            check();
    }
    else
    {
       _clearscreen(_GCLEARSCREEN);
       printa ( "BOARD AVERAGES:", 10, 0, 0);
       for ( dim1 = 0; dim1 < max1; dim1++) {
            check();
         printf ( "\n\ncant # :%d ", (dim1+1) );
            check();
         for ( dim3 = 0; dim3 < max3; dim3++) {
            check();
           if (( UL_X > average [dim1][dim3]) &&
               ( LL_X < average [dim1][dim3] )) {
        printf ( " %8f ", average [dim1][dim3] );
          }
          else
          {
    sprintf (string, " %8f  ", average [dim1][dim3]);
             printa (  string, 10, row(), pos() );
          }
        }
      }
            check();
    }
}
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */


      /* computation for within cant statistics */

void w_cant(void)
{

         float Sw [MAX_DIM1];
         float w_a [MAX_DIM1];
         float w_d [MAX_DIM1][MAX_DIM3];
         float Sa [MAX_DIM1];
```

```
memset (Sw, 0, MAX_DIM1 * sizeof (float));
memset (Sa, 0, MAX_DIM1 * sizeof (float));
memset (w_a, 0, MAX_DIM1 * sizeof (float));
memset (w_d, 0, MAX_DIM1 * MAX_DIM3 * sizeof (float));



          _clearscreen(_GCLEARSCREEN);
          printf ("\n\n\n");
printa ( " within cant between boards std.dev: ", 10,3,1);

          printf ( "\n" );
  for (dim1 = 0; dim1 < max1; dim1++) {
    for ( dim3 = 0; dim3 < max3; dim3++) {
       w_a [dim1] += average [dim1][dim3];
    }
       w_a [dim1] = (w_a [dim1] /max3);
  }


          printf ( "\n\n");
  for ( dim1 = 0; dim1 < max1; dim1++) {
    for ( dim3 = 0; dim3 < max3; dim3++) {
      w_d [dim1][dim3] = average [dim1][dim3] -
                                 w_a [dim1];
      w_d [dim1][dim3] = w_d [dim1][dim3] *
                                 w_d [dim1][dim3];

      Sw [dim1] += w_d [dim1][dim3];
      Sa [dim1] += Sc [dim1][dim3] * Sc [dim1][dim3];
    }
      Sw [dim1] =  Sw[dim1]/max3;
      Sa [dim1] = Sa [dim1]/max3;

      Sw [dim2] = sqrt( Sw[dim1] - (Sa [dim1]/max2));
      printf ( " %8f ", Sw [dim1]);
  }
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

      /* computation for between cant statistics */

void bet_cant(void)
{

        float b_d [MAX_DIM1][MAX_DIM3];
        float b_a [MAX_DIM3];
        float Sb [MAX_DIM3];
        float Sd [MAX_DIM3];
```

```
memset (Sb, 0, MAX_DIM3 * sizeof (float));
memset (Sd, 0, MAX_DIM3 * sizeof (float));
memset (b_d, 0, MAX_DIM1 * MAX_DIM3 * sizeof (float));
memset (b_a, 0, MAX_DIM3 * sizeof (float));


        _clearscreen(_GCLEARSCREEN);
        printf ( "\n\n\n" );
 printa ( " Between cant within saw pocket boards
              std.dev:",10,3,1);
        printf ( "\n\n\n" );
   for ( dim3 = 0; dim3 < max3; dim3++) {
     for ( dim1 = 0; dim1 < max1; dim1++) {
          b_a [dim3] += average [dim1][dim3];
     }
          b_a [dim3] = b_a [dim3]/max1;
   }

   for ( dim3 = 0; dim3 < max3; dim3++) {
     for ( dim1 = 0; dim1 < max1; dim1++) {
       b_d [dim1][dim3] = average [dim1][dim3] -
                                  b_a[dim3];
       b_d [dim1][dim3] = b_d [dim1][dim3] *
                                b_d [dim1][dim3];


       Sb [dim3] += b_d [dim1][dim3];
                Sd [dim3] += Sc [dim1][dim3] *
                                Sc [dim1][dim3];
     }
       Sb [dim3] = Sb [dim3]/max1;
       Sd [dim3] = Sd [dim3]/max1;

   Sb [dim3] = sqrt( Sb [dim3] - ( Sd [dim3]/max2));
       printf ( " %8f ", Sb [dim3]);
   }
}
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

              /* n board statistics */

void n_board(void)
{


         int  num, nb, np, pb;
         char ch, ca, cd, pa, pd;
```

```
                _clearscreen(_GCLEARSCREEN);
printf ( "\n\nDo you want n board averages?
                    type yes or no:");

        scanf( "%s", &ch);
  if ( ch == 'Y' || ch == 'y') {
        printf ( "\n\nDo you want last n cant details?
                    type y or n:");
        scanf("%s", &ca);
    if ( ca == 'Y' || ca == 'y' ) {
        printf ( "\n\nHow many cants? :");
        scanf("%d", &num);
        printf ( "\n\n# of cants = %d\n", num);
      if ( num == 1 )  {
        printf ( "\nDo you want all boards of last cant?
                     type 'y' or 'n':");
        scanf("%s", &cd);
        printf ( "\n\n");
        if ( cd == 'Y' || cd == 'y' ) {
          dim1 = max1 - 1;
          for ( dim3 = 0; dim3 < max3; dim3++) {
            printf ( "%10f", average [dim1][dim3]);
          }
        }
        else
        {
           printf ( "\nHow many boards?: ");
           scanf ("%d", &nb);
           printf ( "\n\n");
           dim1 = max1 - 1;
           dim3 = max3 -nb;
           while( dim3 < max3 ){
             printf ( "%10f", average [dim1][dim3]);
             dim3++;
           }
        }
      }
      else
      {
         printa ( " PRESS ENTER TO CONTINUE", 10, 24, 50);
         while (( inkey = getch()) != CR);
         _clearscreen(_GCLEARSCREEN);
         dim1 = max1 - num;
         while( dim1 < max1 ) {
           check();
           printf ( "\n\ncant # :%d ", (dim1+1));
           check();
           for ( dim3 = 0; dim3 < max3; dim3++) {
```

```
            check();
            printf ( " %8f ", average [dim1][dim3]);
          }
            dim1++;
        }
      }
    }
    else
    {
  printf ("\n\nLast n saw pocket details? type y or n :");
       scanf ("%s", &pa);
      if ( pa == 'Y' || pa == 'y' ) {
        printf ( "\n\nHow many saw pockets? : ");
        scanf ("%d", &np);
        printf ( "\n\n # of pockets = %d\n", np);
        if ( np == 1 ) {
printf ( "\nDo you want all boards of last pocket?
              type 'y' or 'n':");
          scanf( "%s", &pd);
          printf ("\n\n");
          if ( pd == 'Y' || pd == 'y' )  {
            dim3 = max3 -1;
            for ( dim1 = 0; dim1 < max1; dim1++) {
              printf ( "%10f", average [dim1][dim3]);
            }
          }
          else
          {
              printf ( "\nHow many boards? : ");
              scanf ( "%d", &pb);
              printf ( "\n\n");
              dim3 = max3 -1;
              dim1 = max1 - pb;
              while ( dim1 < max1 ){
                printf ( "%10f", average [dim1][dim3]);
                      dim1++;
              }
          }
        }
        else
        {
       printa ( " PRESS ENTER TO CONTINUE", 10, 24, 50);
       while (( inkey = getch()) != CR);
       _clearscreen(_GCLEARSCREEN);
            dim3 = max3 - np;
            while ( dim3 < max3 ) {
                    check();
              printf ( "\n\npoc. # :%d ", (dim3+1));
                    check();
              for ( dim1 = 0; dim1 < max1; dim1++) {
```

```
                    check();
                printf ( " %8f ", average [dim1][dim3]);
              }
                    dim3++;
            }
          }
        }
      }
  }
}
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*                                                               */
/* power.c : Reads power meter data through A/D card.            */
/*                                                               */
/*      AUTHOR   : Fatima Aslam                                  */
/*      VERSION  : V1.0                          901010          */
/*                                                               */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

#include <stdio.h>
#include <stdlib.h>
#include <graph.h>
#include <math.h>
#include <conio.h>
#include <string.h>
#include <dos.h>
#include "bis.c"


void printa ( char *string, int attr, int row, int col );

void CURSOR(int row, int col);
row();
pos();
check();
void graphics_mode(void);
void display_data(void);
void end_program(void);
void card_setup();
void tabulate( void );


#define ANACNT  16
#define DIGCNT  16
#define MAX_SIZE 40
#define CR '\x0d'


unsigned digital[DIGCNT];
float   analog[ANACNT];
struct videoconfig myscreen;
int maxx;
float maxy;
int newx( int );
float newy( float );
int     i;
```

```
main()
{
    card_setup();
    display_data();

}


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

            /* initial setup for A/D card   */

void card_setup()
{

    /* Read CALIB.DAT file and calibrate the analog
                    inputs */

    SH_CALL( "Fn" , analog, digital );   /* Call A-D
                                            driver */

    /* Check if driver & analog card are installed */

    if (digital[0] == 0 && digital[2] == 0)    {
        printf("\nDriver, ADRIVE.COM, not installed; ");
        printf(" or analog card not installed.\n");
        exit(1);
    }

    /* Check if analog card is selected */

    if (digital[0] == 0 && digital[2] != 0)  {
        printf("\nNo analog card selected. BRD SEL
                switch set to 0.\n");
        exit(1);
    }

    /* Check for CALIB.DAT file and FIND.EXE */

    if (digital[0] != 0 && digital[6] == 0)  {
        printf("\nCALIB.DAT file not correct or
                    FIND.EXE was not run.\n");
        exit(1);
    }

    /* Check validity of calibration numbers */

    if (digital[0] > digital[6]) {
        printf("\nCalibration numbers are not
```

```
                    correct.\n");
        exit(1);
    }

    /* Check for proper size of analog & digital
                                channel buffers */

    if (digital[0] > ANACNT ) {
        printf("\nToo many channels installed.\n");
        printf("Change size of ANACNT in heading\n");
        exit(1);
    }

    /* Set analog channel resolution */

    digital[0] = 12;
    SH_CALL( "a" , analog, digital );   /* Call A-D
                                           driver */

    /* Set up analog channel range and calibrate */

    for (i = 0; i < ANACNT; ++i) digital[i] = 4;
    SH_CALL( "rc" , analog, digital );    /* Call A-D
                                             driver */

}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

                /* set graphic mode */

void graphics_mode(void)
{
    _getvideoconfig ( &myscreen );
    _setvideomode ( _HERCMONO );
    _getvideoconfig( &myscreen );
    maxx = myscreen.numxpixels - 1;
    maxy = myscreen.numypixels - 1;
}
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
          /* set horizontal screen  */

int newx( int xcoord )
{
    int nx;
    float tempx;
    tempx = ((float)maxx)/ 500.0;
    tempx = ((float)xcoord) * tempx + 0.5;
    return( (int)tempx );
}
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * */
          /* set vertical screen  */

float newy ( float ycoord )
{
    int ny;
    float tempy;
    tempy = ((float)maxy)/ 100.0;
    tempy = ((float)ycoord) * tempy + 0.5;
    return( (float)tempy );
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * */
           /* end graphic mode */

void end_program( void )
{
     getch();
     _setvideomode( _DEFAULTMODE );
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * */
          /* plot the data */

void display_data(void)
{

       int i,j;
       float UCL,LCL, val1, val2, val3;
       float reference = 0.0;
       float y_change, y, y1;
       char c, inkey;
       char fh, sv;



        UCL = 0.0;
        LCL = 0.0;

   /* Enter the control limits & reference value */

    _clearscreen(_GCLEARSCREEN);
 printf ( " Do you want to change the control limits,
                Enter 'y' or 'n' :");
    scanf ("%s", &sv);
    if ( (sv  == 'Y') || (sv == 'y')) {
      printf ( "\n\n\n Enter Control limits : " );
      scanf ( "%f%f", &val1, &val2 );
      printf ( "\n\n\n Reference level : " );
```

```
      scanf ( "%f", &val3 );
    }

      UCL = val1;
      LCL = val2;
      reference = val3;
      printf ( "\nUpper control limit = %f\n", UCL );
      printf ( "\nLower control limit = %f\n", LCL );
      printf ( "\nReference level = %f\n", reference );
      printf ( "\n\n\nPress key 1 to enter :");

    fh = getch();
    printf ( "%c", fh);
if (fh == '1')
{
     printf ( "\n\nDo you need a plot? :
                    type 'Y'es or 'N'o ->");
     c = getch();
     printf ( "%s", c);
     _clearscreen(_GCLEARSCREEN);

   if ( c == 'Y' || c == 'y' )  {
        graphics_mode();
      _setvieworg( 0, newy( 50.0 ));
      _setlinestyle( 0x5555 );
      _moveto( 0, 0 );
      _lineto ( 0, newy( 50.0 ));
      _moveto( 0, 0);
      _lineto ( 0, newy( -50.0 ));
      _moveto( 0, 0);
      _lineto (newx(500), 0);
      _moveto( 0, 0);


      _setlinestyle( 0xFFFF);
      y = UCL-reference;
      _moveto (0,newy(-y));
      _lineto (newx(500),newy(-y));
      _moveto ( 0, 0);
      y1 = reference - LCL;
      _moveto (0,newy(y1));
      _lineto (newx(500),newy(y1));
      _moveto (0,0);



    printf ( "\nUpper control limit = %f\n", UCL );
    printf ( "Lower control limit = %f\n", LCL );
    printf ( "Reference level = %f\n", reference );
    j = 0;
```

```
    _setlinestyle( 0xFFFF);
    for( i = 0; i < MAX_SIZE; i++)    {

      /* Calibrate and read analog inputs */

      SH_CALL( "cm" , analog, digital );

 analog[0] = -((600 * analog[0] )/(.993 * 1.4142));
      analog[0] -= reference;
      y_change = newy(-analog[0]);
      j++;
      _lineto(newx(j*10), (y_change) );

      if( (j*10) == 500 )      {
        j = 0;
        _clearscreen(_GCLEARSCREEN);
        printf ( "\nUpper control limit = %f\n", UCL );
        printf ( "Lower control limit = %f\n", LCL );
        printf ( "Reference level = %f\n", reference );
        _setvieworg( 0, newy( 50.0 ));
        _setlinestyle( 0x5555 );
        _moveto( 0, 0 );
        _lineto ( 0, newy( 50.0 ));
        _moveto( 0, 0);
        _lineto ( 0, newy( -50.0 ));
        _moveto( 0, 0);
        _lineto (newx(500), 0);
        _moveto( 0, 0);


        _setlinestyle( 0xFFFF);
        y = UCL-reference;
        _moveto (0,newy(-y));
        _lineto (newx(500),newy(-y));
        _moveto ( 0, 0);
        y1 = reference - LCL;
        _moveto (0,newy(y1));
        _lineto (newx(500),newy(y1));
        _moveto (0,(y_change));
      }
     }
         end_program();
         _displaycursor( _GCURSORON);
   }
   else
   {
       tabulate();
   }
 }
}
```

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
          /* Tabulate the data */

void tabulate( void )
{

     int i,j;
     float UCL,LCL, val1, val2;
     char sh;
     char string[128];


     UCL = val1;
     LCL = val2;

     printf ("Do you want 'BEEP':type 'Y'es or 'N'o ->");

     sh = getch();
     printf("%s", sh);

 if (sh == 'Y' || sh == 'y'){
     _clearscreen(_GCLEARSCREEN);
     printa ( "Power meter reading:",10,0,0);
     printf("\n\n");
   for( i = 0; i < MAX_SIZE; i++)  {
         check();
     SH_CALL( "cm" , analog, digital );
     analog[0] =-( (600 * analog[0] )/(.993 * 1.4142));

     if ( (analog[0] < UCL)  && (analog[0] > LCL) ) {
       printf ( "%16f", analog[0]);
     }
     else
     {

       printf( "\a");
       sprintf(string, "%16f ", analog[0]);
       printa ( string, 10, row(), pos());
     }
   }
 }
 else
 {
   _clearscreen(_GCLEARSCREEN);
   printa ( "Power meter reading:",10,0,0);
   printf("\n\n");
   for( i = 0; i < MAX_SIZE; i++) {
       check();
     SH_CALL( "cm" , analog, digital );
```

```
      analog[0] =-( (600 * analog[0] )/(.993 * 1.4142));
      if ( (analog[0] < UCL)  && (analog[0] > LCL) ){
        printf ( "%16f", analog[0]);
      }
      else
      {
        sprintf(string, "%16f ", analog[0]);
        printa ( string, 10, row(), pos());
      }
         check();
    }
  }
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
```