WELDSMART:
A VISION-BASED EXPERT SYSTEM
FOR QUALITY CONTROL

Final Report

Mid-South Engineering, Inc.
2131 Belcourt Ave
Nashville, TN 37212

Mid-South Engineering, Inc.: Kristinn Andersen
Robert Joel Barnett
James F. Springfield

Vanderbilt University, Subcontract: Dr. George E. Cook

NASA Contract NAS8-37685

September, 1992

TABLE OF CONTENTS

Page

LIST OF TABLES iv

LIST OF FIGURES v

LIST OF TERMS viii

Chapter

I. EXECUTIVE SUMMARY 1

II. OBJECTIVES AND SCOPE OF THE WORK 2

Introduction and Objectives 2
Main Results and Suggestions for Further Implementations 4

III. BACKGROUND - EXISTING TECHNOLOGIES 5

Visual Inspection and Other Inspection Methods 5
Image Processing and Analysis Techniques 8
Image Digitization 9
Image Preprocessing 10
Image Analysis and Feature Extraction 12

IV. IMAGE ANALYSIS AND EVALUATION FOR QUALITY CONTROL 14

Feature Enhancement 14
Analysis Using Artificial Neural Networks 17
Rule-Based Expert Systems 18

V. QUALITY EVALUATION 21

Quality Evaluation for the Gas Tungsten Arc Welding Process 23
Quality Evaluation for the Gas Metal Arc Welding Process 38
Quality Evaluation for the Variable Polarity Plasma Arc Welding Process 52
The Integrated Weld Quality Evaluation Approach 58

VI. QUALITY CONTROL SYSTEM PROTOTYPE IMPLEMENTATION 61

Hardware Description 61
Software Description 62

VII. CONCLUSIONS AND SUGGESTIONS FOR COMMERCIALIZATION 70

BIBLIOGRAPHY 71

APPENDIX A: Artificial Neural Networks 73

Historical Overview 74
The Backpropagation Network 75
Neural Network Applications 79
Neural Network Hardware and Speed Considerations 80

APPENDIX B: Source Code Listing for the Quality Analysis System 82

APPENDIX C: Source Code Listing for Device Drivers 124

LIST OF TABLES

Table Page

1. Sample results obtained with the weld profile analysis neural network. These six weld profiles were not used in original training of the network. 58

LIST OF FIGURES

Figure Page

1. An unprocessed image showing a crack emerging from a machine part. 15
2. An image of the crack in the workpiece after magnetic particle material has been applied. 16
3. The image of the crack after edge detection algorithm has been applied. 17
4. A schematic diagram of the CLIPS rule-based expert system. 19
5. An overview of the quality evaluation process. 22
6. A photograph of a good quality GTA weld bead. 24
7. A photograph of a low quality GTA weld bead. 24
8. The horizontal integration procedure illustrated. This technique is used in analysis of GMA weld quality. 26
9. Image scanning data representing a GTA weld profile. 27
10. Image scanning data representing a GTA weld profile, filtered to aid further computer analysis. 27
11. The vertical integration procedure illustrated. This technique is used in analysis of GTA weld quality. 28
12. The vertical integration procedure illustrated. This technique is used in analysis of GTA weld quality. 29
13. An overview of the process developed for evaluating GTA weld quality. 30
14. Scanned ripple pattern of weld GTAW2 (human quality evaluation = 2). 32
15. Filtered ripple pattern of weld GTAW2 (human quality evaluation = 2). 32
16. Scanned ripple pattern of weld GTAW4B (human quality evaluation = 4). 33

17. Filtered ripple pattern of weld GTAW4B (human quality evaluation = 4). 33

18. Scanned ripple pattern of weld GTAW6 (human quality evaluation = 6). 34

19. Filtered ripple pattern of weld GTAW6 (human quality evaluation = 6). 34

20. Scanned ripple pattern of weld GTAW9B (human quality evaluation = 9). 35

21. Filtered ripple pattern of weld GTAW9B (human quality evaluation = 9). 35

22. Comparison of the calculated quality indicator and the corresponding human quality evaluations. 37

23. A photograph of a good quality GMA weld bead. 39

24. A photograph of a poor quality GMA weld bead. 39

25. An overview of the process developed for evaluating GMA weld quality. 41

26. Horizontal integration scan No. 1 of weld GMA6. 42

27. Horizontal integration scans of weld GMA2. 43

28. Horizontal integration scans of weld GMA4. 43

29. Horizontal integration scans of weld GMA6. 44

30. Horizontal integration scans of weld GMA8. 44

31. Horizontal integration scans of weld GMA9. 45

32. The envelope difference of scans across weld GMA2 (human quality evaluation = 2). 46

33. Root-mean-square variance of scans across weld GMA2 (human quality evaluation = 2). 46

34. The envelope difference of scans across weld GMA4 (human quality evaluation = 4). 47

35. Root-mean-square variance of scans across weld GMA4 (human quality evaluation = 4). 47

36. The envelope difference of scans across weld GMA6 (human quality evaluation = 6). 48

37. Root-mean-square variance of scans across weld GMA6 (human quality evaluation = 6). 48

38. The envelope difference of scans across weld GMA8 (human quality evaluation = 8). 49

39. Root-mean-square variance of scans across weld GMA8 (human quality evaluation = 8). 49

40. The envelope difference of scans across weld GMA9 (human quality evaluation = 9). 50

41. Root-mean-square variance of scans across weld GMA9 (human quality evaluation = 9). 50

42. The GMAW computer quality indicator as a function of the human-assigned quality values. 51

43. The VPPAW process and the components used to scan and analyze the weld profile. 53

44. A photograph of the cross section of a typical VPPA weld. 54

45. The procedure for VPPA weld evaluation. 55

46. A neural network that locates the crown, undercuts and weld boundaries from weld profile data. 56

47. A scanned weld profile, with the weld crown, undercuts, and parent metal boundaries indicated by the neural network and by the traditional system. 57

48. Successive laser scans across a VPPAW root pass may be used to reconstruct a view of the three-dimensional weld bead surface. 59

49. An overview of an integrated weld evaluation system. 60

50. The processing element of a neural network. 75

51. A backpropagation neural network. 76

LIST OF TERMS

AET	Acoustic Emission Testing
AI	Artificial Intelligence
ANN	Artificial Neural Network
CCD	Charge Coupled Device
CID	Charge Injection Device
DT	Destructive Testing
ET	Eddy Current Testing
GMAW	Gas Metal Arc Welding
GTAW	Gas Tungsten Arc Welding
HSI	Hue-Intensity-Saturation
MOSFET	Metal-Oxide Semiconductor Field Effect Transistor
NDE	Non-Destructive Evaluation
NDT	Non-Destructive Testing
NTSC	National Television Standard Committee
MS-DOS	Microsoft Disk Operating System
MT	Magnetic Particle Testing
PAN	Parallel Associative Network
PT	Liquid Penetrant Testing
RGB	Red-Green-Blue
RT	Radiographic Testing
UT	Ultrasonic Testing
VGA	Visual Graphics Array

VLSI	Very Large Scale Integration
VPPAW	Variable Polarity Plasma Arc Welding
VT	Visual Testing

CHAPTER I

EXECUTIVE SUMMARY

This work was aimed at exploring means for utilizing computer technology in quality inspection and evaluation. Inspection of metallic welds was selected as the main application for this development and primary emphasis was placed on visual inspection, as opposed to other inspection methods, such as radiographic techniques. Emphasis was placed on methodologies with the potential for use in real-time quality control systems.

Because quality evaluation is somewhat subjective, despite various efforts to classify discontinuities and standardize inspection methods, the task of using a computer for both inspection and evaluation was not trivial. The work started out with a review of the various inspection techniques that are used for quality control in welding. Among other observations from this review was the finding that most weld defects result in abnormalities that may be seen by visual inspection. This supports the approach of emphasizing visual inspection for this work.

Quality control consists of two phases: (i) identification of weld discontinuities (some of which may be severe enough to be classified as defects), and (ii) assessment or evaluation of the weld based on the observed discontinuities. Usually the latter phase results in a pass/fail judgement for the inspected piece. It is the conclusion of this work that the first of the above tasks, identification of discontinuities, is the most challenging one. It calls for sophisticated image processing and image analysis techniques, and frequently *ad hoc* methods have to be developed to identify specific features in the weld image. The difficulty of this task is generally not due to limited computing power. In most cases it was found that a modest personal computer or workstation could carry out most computations in a reasonably short time period. Rather, the algorithms and methods necessary for identifying weld discontinuities were in some cases limited.

The fact that specific techniques were finally developed and successfully demonstrated to work illustrates that the general approach taken here appears to be promising for commercial development of computerized quality inspection systems. Inspection based on these techniques may be used to supplement or substitute more elaborate inspection methods, such as x-ray inspections.

CHAPTER II

OBJECTIVES AND SCOPE OF THE WORK

The main objective of this work was to explore means to incorporate computer technology into the process of quality inspection and evaluation, particularly for welding applications. The commercial value of such work is of great importance, and thus the general approach was oriented towards using technology that was affordable and readily available. Visual inspection was selected as the primary means for quality assessment, as it has relatively modest equipment requirements and a large portion of defects may be indicated by visual inspection.

Introduction and Objectives

The role of quality control in the manufacturing processes has become increasingly important as national and global competition has intensified in recent years. Through technological advances, such as digital computing and signal processing, improvements in quality control can be implemented in two key areas. First, the process of quality control can be greatly enhanced through the use of digital computers for data analysis and records keeping, and through sophisticated processing of sensory signals such as acoustic signatures or digitized images. Secondly, technological developments can be utilized to automate the quality inspection process to a significant degree, and thus make it more efficient and cost effective.

Quality control is an essential part of any manufacturing process as it provides the manufacturer and the user of the manufactured item assurance of performance and reliability to a specified degree. Expensive failures in large projects can frequently be avoided by relatively inexpensive preventive measures in quality control. Thus, efforts spent on quality control are frequently well worth their cost.

While most aspects of the manufacturing processes have progressed substantially with advances in robotics, electronics, and related fields, the crucial task of quality control still relies in most cases on traditional approaches of human judgement. Although human quality judgement has served manufacturers quite adequately for a long time its drawbacks and limitations have emerged as industrial competition has intensified, complexity of manufactured products and systems has increased, and quality requirements and performance specifications have become stricter. A number of specific improvements can be accomplished by automating the quality evaluation process, as discussed in the following paragraphs.

Consistency and reliability of the inspection can be improved. It is well known that human inspectors can be fairly inconsistent, both individually as a person's judgement criteria varies from time to time, and collectively as one person's judgments may vary from those of another one. Although rigid quality inspection and manufacturing standards are imposed to ensure adequate product performance such schemes will only be as reliable as the people enforcing them.

Focus on details to an extent unsurpassable by human inspectors is possible in automatic systems. While humans usually judge quality by overall appearance and at best keep alert against specific details commonly encountered in the manufacturing process, a non-human quality controller may be programmed for analysis to any specified level of detail.

Reasoning leading to each judgement and references to previous judgments are easily accessible. There will never be doubt to as why a computerized quality inspector rejects or accepts an item. All reasoning is performed according to known rules or calculations. The information on which a judgement is based is archived so it may be accessed at later times by humans or by the computer for its own use and reference.

Speed of evaluation is frequently a concern. By employing appropriate hardware and software structures the speed of an automatic quality inspector can surpass that of a human inspector.

Cost, the bottom line, is always significant in manufacturing processes. While it can be argued that initially the cost of a computerized quality controller will surpass those of human inspectors the cost ratio will be reversed as time proceeds.

The level of automation in the computerized quality control system may vary, depending on the application. In the ideal situation the proposed system would be capable of inspection and evaluation in a completely autonomous manner. However, a more realistic approach has been taken here in developing a prototype system which tackles a specific subset of the entire problem of quality control. Furthermore, situations will arise when human interceptions are required to evaluate the examined items. The system is therefore designed so as to facilitate human interactions. In addition to using the techniques developed for this work for actual quality control tasks, the system may also be used as a training tool for human quality inspectors.

The work carried out during the contract period is to be general enough to benefit a variety of quality control applications at NASA, in addition to commercial applications, particularly those relating to materials inspection and evaluation.

Main Results and Suggestions for Further Implementations

The results of this work strongly suggest that visual inspection procedures, that so far have mainly been carried out by human experts, may at least to some extent be implemented on microcomputers. Quality control has traditionally, and for good reasons, been a human-dominated field. Many of the inspection tasks and judgments that quality control requires is not easily specified or implemented in computer terms, which has left this field to a large degree outside the mainstream of computer applications. The rapid development of computer technology in recent years, however, has been narrowing this gap between human and automatic inspection.

This report outlines some of the avenues that were explored during the contract and the main results. It is shown that a variety of image processing techniques exist to enhance and extract certain features from weld images. Furthermore, specific methods are illustrated that assign numeric quality indicators to welds, based on their visual appearance.

The work was carried out using readily available computer and imaging equipment. The main results consist of two parts, both of which are delivered to NASA at the conclusion of the contract. First, the applied research and experimentation that was carried out to accomplish the objectives of the work are summarized in this report. Secondly, a software package was developed that may serve as a template for a commercial visual inspection system. This software package is delivered to NASA, as well.

A proposed continuation of this work will result in a commercial system for quality inspection. The software will be modified based on the experience obtained with the prototype system, and the hardware will continue to consist of readily available components.

CHAPTER III

BACKGROUND - EXISTING TECHNOLOGIES

A number of techniques are presently widely used for quality inspection of metal fabrication and welding in particular. A brief overview of these methodologies is given in the following paragraphs.

Visual Inspection and Other Inspection Methods

While a number of methods are currently used for quality inspections, each one has its own area of applicability, advantages, and disadvantages. An extensive survey of the state-of-the-art of available methods is vital before the vast task of designing and building a comprehensive quality evaluation system is initiated. By studying existing testing methods a thorough comparison of the various approaches can be carried out. The comparison is to illustrate the areas of applicability for each of the testing methods, specific strengths, and limitations. This study should enable the system designers to program the system to select the optimal testing methods for each test situation.

The testing methods have traditionally been categorized as nondestructive tests (NDT) or nondestructive evaluation (NDE) on one hand and destructive tests (DT) on the other. As the term indicates NDT methods are those testing methods which convey information on discontinuities in the inspected item or material without affecting any of its characteristics for usability. This is contrasted to the DT methods which require the tested sample to be affected to the extent that it can not be put to its intended use after testing. Clearly the NDT methods are preferable to DT whenever applicable. This holds particularly for manufacturing of small-quantity items and wherever production price is relatively high, due to high materials cost, extensive precision, etc. On the other hand DT frequently yields information which may not be attainable by NDT.

Beyond the DT-NDT classification the testing methodologies can be categorized according to which defect types they emphasize. Many defects can be detected by more than one testing technique and with varying degrees of accuracy. Therefore there is a considerable overlap of applicability of the various testing methods. By a comprehensive study of the available testing methodologies this research would enable selection of a well chosen set of testing techniques which would allow quality inspection for any application in an efficient manner. For further clarifications some of the most commonly used quality testing methods will be reviewed here.

Visual Testing (VT) [1] is the most readily applicable testing method for human inspectors. Its primary advantage is its relative simplicity as far as equipment requirements is concerned. In the simplest applications the inspector uses relatively simple tools, such as rulers, micrometers and optical comparators to classify and quantify the defects, in addition to a light source and magnifying glasses. These tools are, for example sufficient for inspection of many surface defects in metals. There are mainly two limitations to this test: First, it only conveys defects appearing on the surface of the inspected object, and secondly, it depends heavily on the visual acuity of the quality inspector. The types of defects frequently appearing on the examined metal or ceramic surface include cracks, microfissures and sometimes porosity. On welds common defects include overlaps, undercuts, unfilled craters, and pock marks. Besides defects directly related to joint itself, visual inspection can manifest other deficiencies, such as incorrect joint alignment and structure distortion caused by heat, mechanical pressure, or other causes of deformation.

Radiograph Testing (RT) [2] utilizes radiation which penetrates the inspected object to allow examination of defects internal to the material. The most common radiation types are x-rays and gamma rays. The radiation source is placed on one side of the material and a radiation-sensitive detector (a photosensitive film or an electronic scanning device) on the other. The photographic approach is the most common one, which requires the exposed film to be developed and then interpreted by a trained quality inspector. The details of the equipment commonly applied for this task and the exact procedures for optimizing image quality will not be dwelled upon here. Instead, the applicability and limitation are briefly discussed. The radiographic technique identifies most types of material defects, internal as well as external. For fusion-joined materials typical defects identifiable by radiographs are cracks, porosity, lack of fusion, incomplete penetration, and slag, as well as corrosion, fit-up defects, wall thickness and dimensional distortions. The price for such generality, however, can be substantial. The equipment needed for radiographic inspections consists of radiation sources (x-ray or gamma ray sources with appropriate projecting apparatus), films and film holders, film processing facilities, and radiation monitoring equipment. Interpreting the radiographs calls for highly trained and skilled personnel. Finally, the radiation is potentially hazardous and therefore the exposure requires care and a skilled operator.

Magnetic Particle Testing (MT) [3] is based on the principle that discontinuities in a ferromagnetic piece of metal break up magnetic flux in the metal and produce localized magnetic poles at the discontinuities. Magnetic flux is conducted through the inspected metal object by placing it in a strong magnetic field from an external magnet or by conducting current through the metal. Magnetic particles are spread over the surface of the metal and they form patterns along the magnetic flux lines. If discontinuities occur at or near the surface they disrupt the magnetic flux in the tested object. The disruptions, in turn, reorient the patterns of the magnetic particles and thus indicate the defect locations. The equipment needed for magnetic particle testing includes magnetizing coils or current prods, a current source, magnetic powder, and sometimes ultraviolet light to examine the magnetic patterns. While this method is relatively economical, it is limited to ferro-magnetic materials and defects located at or near the surface.

Liquid Penetrant Testing (PT) [4] uses the fact that certain liquids can enter voids and cracks and remain there when the surface liquid is removed. Thus, these liquid residuals can reveal even the smallest imperfections on the material surface. The liquids used for this purpose are either visible in ordinary white light, or in ultraviolet light. The testing procedure is simple and requires only the fluorescent or dye penetrant liquid, developer and cleaning chemicals and optionally a fluorescent light source. The main limitation is that this testing method is only applicable to surface voids, such as surface cracks, microfissures, and porosities.

Eddy Current Testing (ET) utilizes the principle that an electric current will flow in any conductor subjected to a changing magnetic field. Application of a varying magnetic field to a metal object generates electric currents in the metal, whose conductivity usually decreases as the number and size of discontinuities increase. An inductor, carrying alternating current, is usually employed for generating the magnetic field. Resistance to the eddy currents is reflected into the magnetic field and this affects the impedance of the inductor. Thus, the internal discontinuities of the metal can be detected by monitoring the inductor impedance. Furthermore, the frequency of the inductor current influences the sensitivity of this technique to various defect types. The main equipment required for the testing includes an induction coil, a current source, magnetic pickup coils, current meters, and calibration standards. While the test equipment is usually fairly portable, the method is usually limited to surface and shallow discontinuity detection.

Ultrasonic Testing (UT) [5] involves directing a high frequency sound beam into the test material on a predictable path, which, upon reflecting back from a discontinuity in the material, produces a wave that is amplified and usually displayed on a graphic device. Thus, the underlying principles are the same as used for radar, sonar, and related echo detecting techniques. The ultrasonic waves or pulses are generated by electronic means and transmitted to one or more piezoelectric devices which transform the electrical energy into mechanical waves. The piezoelectric transducers are attached to the tested metal, frequently on a thin layer of liquid, which serves as a coupling medium for the acoustic signals. Detection is carried out through similar piezoelectric transducers which convert the received sound waves into electrical signals. These signals are displayed on graphic devices, such as cathode ray tubes. Principally this testing method is very powerful as it can convey most defect types and is relatively portable. On the other hand, it requires skilled personnel for interpretation of the detected signals.

Acoustic Emission Testing (AET) [6,7,8] is still considered to be in its early stages of industrial use. It consists of the detection of acoustic signals produced by plastic deformations or crack formation during loading. Detection is, as in the case of ultrasonic testing, carried out by piezoelectric transducers. These signals are amplified, passed through signal processing equipment, and interpreted by their appearance on a graphic device. AET is useful for continuous surveillance inspection, and detection of cracking in loaded structures and during material cooling. Fairly sophisticated signal processing and interpretation skills are required for this testing technique.

Of the various testing techniques summarized above, this work has primarily been concerned with testing based on visual inspection. As illustrated later in this report, images of weld surfaces may be used to evaluate and assess weld quality. Image inspection was used as the means for judging the quality of welds from the GTAW and GMAW processes. In principle, the same approach may be taken to examine VPPA welds. However, the section on VPPAW inspection, later in this report, illustrates the use of a laser profile scanner, which may be used to provide information on weld surface conditions as well.

Image Processing and Analysis Techniques

A summary of image digitizing, analysis and processing techniques will be given in the following subsections. This summary should familiarize the reader with the approaches discussed relating to this work as well as the terminology involved.

The concept of "computer vision" has been separated into three distinct tasks. Image digitizing is the first step, where the inspected item is captured by a camera and digitized for further digital processing. The second task is preprocessing of the image, which generally serves to improve it, eliminate irrelevant or unwanted artifacts, and enhance the features that are of importance for the task, i.e., quality control. The third step in the computer vision process is analysis of the image, which ultimately results in conclusions relating to the features found in the image, such as discontinuities or defects in the examined part.

Image Digitization

A number of devices exist for capturing images and converting them to electronic form for further processing. The light path of most such devices follows that of regular cameras, but instead of using a chemical film plane the electronic image digitizer uses sensors which convert incident light intensity into electrical signals.

Image sensors based on vacuum tube technology include the image orthicon tube, the vidicon tube, the iconoscope tube and the image dissector tube. The orthicon tube is a high-quality device in that it offers very stable performance at all incident light levels, and because of its qualities it is widely used in commercial television. The vidicon is smaller and more rugged than the orthicon, and thus it is better suited for portable operations and applications where small size is desirable. The iconoscope tube is now mostly of historical interest, and the image dissector tube is not widely used either.

Solid-state sensors are rapidly taking over the functions that vacuum tube image sensors used to perform only a few years ago. Broadly, the solid-state image sensors can be divided into two subgroups: Charge-coupled devices (CCD's) and charge injection devices (CID's). Both techniques are based on arrays of metal-oxide field effect transistors (MOSFET's), in which incident photons generate electrical charges that may be sensed and amplified. In the case of CCD's each row in the two-dimensional array is configured as a shift register with a single output. A series of clock pulses is applied to this register and the photon-induced charges are read sequentially from the register output. In contrast to this, the CID's retain the electrical charge in its fixed place in the array and the amount of this charge is accessed by addressing signals on the semiconductor array. The camera used for this work is based on the CCD principle, and the remainder of this discussion will be centered on this technology.

The resolution of the digitized image depends on the size of the CCD array. Usually the array is square (i.e., the number of rows equals the number of columns) and typically the number of either rows or columns is an integer power of 2, such as 256 or 512. Thus, the digitized image, whether obtained through a CID or a CCD, may be viewed as a matrix of individual sensor elements, pixels, each of which holds a voltage which is related to the incident light intensity. The update rate of the pixel values can generally be controlled electronically, and is typically set at 60 Hz (the U.S. television system screen refresh rate).

For digital processing of the image the voltage value of each pixel is digitized to a numerical value. This is usually accomplished with electronic hardware referred to as image digitizers or frame grabbers. The image digitizer converts each pixel in the image to a numerical value which is a function of the brightness or the color of the pixel. In the case of color images each pixel may characterized by three numbers, each one indicating the intensity of a primary color in that pixel: red, green and blue. A white pixel has equal amount of the three primary colors, and its brightness is determined by the equal values of these colors. For example, a given pixel value may be represented by three color registers in the image digitizer, each of which may contain numbers between 0 and 255. With all three register containing 255 the resulting pixel color may be bright white, with all three register storing the value 127 the color is gray, and with 0 in all register the pixel is black. Any color combination can be obtained by mixing the color values in appropriate proportions. For example, a green color can be obtained by assigning relatively high numbers to the blue and the yellow registers while leaving the red register at a low value or zero. An alternative to the red-green-blue (RGB) system is another one in which the pixel color attributes are stored as "hue", "saturation" and "intensity" (HSI). The intensity component determines the overall brightness of the pixel. The saturation is the purity of the pixel color. Highly saturated colors are vivid while low saturation results in dull or faded colors. Finally, the hue is the actual color. Using either of the above colors schemes, an entire color image can be stored in digital memory, where it may be used for further processing. Frequently the image digitizer is also capable of converting the digital signals of the stored image to analog signals which may be routed to a monitor for viewing.

Image Preprocessing

The first step in processing a digitized image is generally referred to as preprocessing, which encompasses a number of operations. The purpose of preprocessing is usually to enhance general attributes of the image, such as its contrast, average intensity, and thus the preprocessing methods typically operate on the entire picture.

Equalization is an image processing technique, which serves to enhance contrasts in the digitized image and neutralize the effects of excessive or inadequate lighting. This is useful in real-time welding operation where the light illuminating the workpiece is frequently dominated by the arc brightness, which is not readily controllable for light intensity. In one relatively simple approach to equalization the image is scanned and the maximum and minimum pixel brightness values (intensity, in the case of color images) are established. The intensity minimum and maximum observed in the image may or may not reach the minimum and maximum that the image digitizer offers (e.g., 0 and 255, respectively). Once the original pixel intensity range has been determined a new intensity is assigned to each pixel so that the entire digitizer range is utilized. Specifically, if p_{min} and p_{max} are the initial minimum and maximum intensities in the scanned image, each intensity, p, will be updated to a new intensity value, q, according to the following linear transformation:

$$q = q_{max} \, (p-p_{min})/(p_{max}-p_{min}) \qquad (1)$$

where q_{max} is the maximum pixel intensity level supported by the image digitizer, while its minimum intensity level is assumed to be zero. The result of this operation is an image that generally has stronger contrasts and is less sensitive to improper lighting than the original image.

Various filtering techniques constitute a host of preprocessing tools. Generally, filtering can be considered as a means for transforming the image intensities in some way so as to enhance or deemphasize selected features.

Smoothing or averaging is a relatively simple method for reducing the effects of individual pixel abnormalities or "noise" in the image. This operation is essentially equivalent to low pass filtering of sequential signals, except in this case the signal is the image, which is two-dimensional. With this technique each pixel in the image is assigned a new brightness level which is based on its original brightness as well as those of the pixels surrounding it. For example, the brightness of each pixel in the image may be set as the average of the 3-by-3 array of pixels in which it is centered. When this has been repeated for each pixel, the resulting image is generally more free of noise

A more sophisticated method of equalization than the one described earlier in this section is the histogram equalization. The pixels of the image are ordered according to brightness values and a histogram of the encountered levels is constructed. A bright image will generally reveal large histogram values at high brightness values and vice versa for a dark image. Alternatively, a dull image contains small histogram values at the two brightness extremes while a high contrast image reveals larger numbers at very high and very low brightnesses. Through histogram equalization the brightness levels found in the original image are mapped so that a histogram of the image after the mapping would result in equal number of all brightnesses. The original image may produce a histogram $h(p)$ and the histogram of the equalized image may be denoted as $g(q)$, where p and q represent brightness levels. Clearly, the number of any given set of original pixels with brightness in the range p to $p+dp$ equals the number pixels with corresponding brightnesses between q and $q+dq$ after the histogram mapping. In other words:

$$h(p)\ dp \quad = \quad g(q)\ dq \tag{2}$$

Furthermore, if the new histogram, $g(q)$, is to be uniform, then:

$$g(q)\ dq \quad = \quad N^2/M \tag{3}$$

where N is the number of pixels in the image and M is the number of brightness levels. The histogram equalization transform is therefore found as:

$$q(p) \quad = \quad (M/N^2)\sum_{0}^{p} h(s)\ ds \tag{4}$$

Image Analysis and Feature Extraction

Template matching is a relatively simple operation that facilitates the detection of a specified feature in the image. The feature that is sought must be well defined and known in terms of dimensions and brightness levels. Usually template matching is accomplished through correlation. Basically, the correlation procedure is carried out as follows. The sought feature, e.g., a specified item in an image, is defined on a grid of pixels, a template, that may be moved around the analyzed image. For each location of the template in the image the difference between each template pixel brightness and the corresponding image pixel is calculated, squared, and the sum of all such squares over the entire template is determined. This sum is an indicator of the overall difference between the template and the part of the image located under the template. Specifically, the sum is expressed as:

$$D(\underline{y}) = \sum_{\underline{x}} [f(\underline{x})-t(\underline{x}-\underline{y})]^2 \tag{5}$$

Here, $f(\underline{x})$ is the image, which consists of pixels $\underline{x} = (x_1,x_2)$, and $t(\underline{x}-\underline{y})$ is the template, displaced to location $\underline{y} = (y_1,y_2)$. The sum $D(\underline{y})$ varies with $\underline{y}$ and is taken over all of the template pixels. The squared term may be expanded as:

$$[f(\underline{x})-t(\underline{x}-\underline{y})]^2 = f^2(\underline{x}) - 2f(\underline{x})t(\underline{x}-\underline{y}) + t^2(\underline{x}-\underline{y}) \tag{6}$$

Clearly, $f^2(\underline{x})$ is independent of $\underline{y}$, and thus it is constant. Similarly, the template, $t^2(\underline{x}-\underline{y})$, is by definition fixed, which leaves the center term as the variable component. The variable component is proportional to the cross correlation between $f(\underline{x})$ and $t(\underline{x}-\underline{y})$, which is defined as:

$$R_{ft}(\underline{y}) = \sum_{\underline{x}} [f(\underline{x})\ t(\underline{x}-\underline{y})] \tag{7}$$

and is maximized when the image under $t()$ is identical to $t()$ itself. Thus, correlation can be used to search for a known item or feature in the image and determine its location as well.

Edge detection is an operation that is concerned with locating areas of abrupt brightness variations in the image, such as at edges or boundaries between objects. A simple edge detection operator is one that replaces each pixel with the magnitude of the brightness gradient at that pixel. For example, each pixel brightness $p(x)$, for a pixel located at $x=(x1,x2)$, may be replaced by the quantity $G(x)$, determined as:

$$G^2(x) = [p(x_1+1,x_2)-p(x_1,x_2)]^2 + [p(x_1,x_2+1)-p(x_1,x_2)]^2 \tag{8}$$

Edge detection is useful for enhancing artifacts in the image, such as cracks or other sharp features. Edge detection needs to be used carefully, however, as it tends to highlight arbitrary noise pixels in the image, as well.

CHAPTER IV

IMAGE ANALYSIS AND EVALUATION FOR QUALITY CONTROL

While the previous chapter outlined some of the general aspect of image processing and common image processing techniques, this chapter will be devoted to specific techniques relevant to image processing for quality control.

As stated earlier, evaluation of metallic welds has been selected as a specific quality control application for this work, although other types of materials inspection may benefit from the work as well. Evaluation of weld quality takes a number of factors into account and is, when taken to its ultimate degree, a fairly sophisticated process. In this work a number of specific features of the weld bead were selected to represent the overall weld quality. The set of examined features may not always contain every possible defect type, but some of the most common characteristics separating "good" welds from the "bad" ones were examined for this purpose.

Feature Enhancement

Most visual weld defects are characterized as sharp artifacts in the image. This includes cracks, weld bead ripple patterns, spatter and porosities. Edge detectors were therefore important tools for enhancing these weld features.

The gradient operator, described earlier, was used for most edge detection tasks. Figures 1 through 3 illustrate examples of edge detection applications. Figure 1 shows a metallic piece with a crack propagating from its edge. In figure 2 the crack has been enhanced by a magnetic particle test. Applying the edge detection operation on this image highlights all sharp features, resulting in a much enhanced view of the crack, as shown in figure 3.

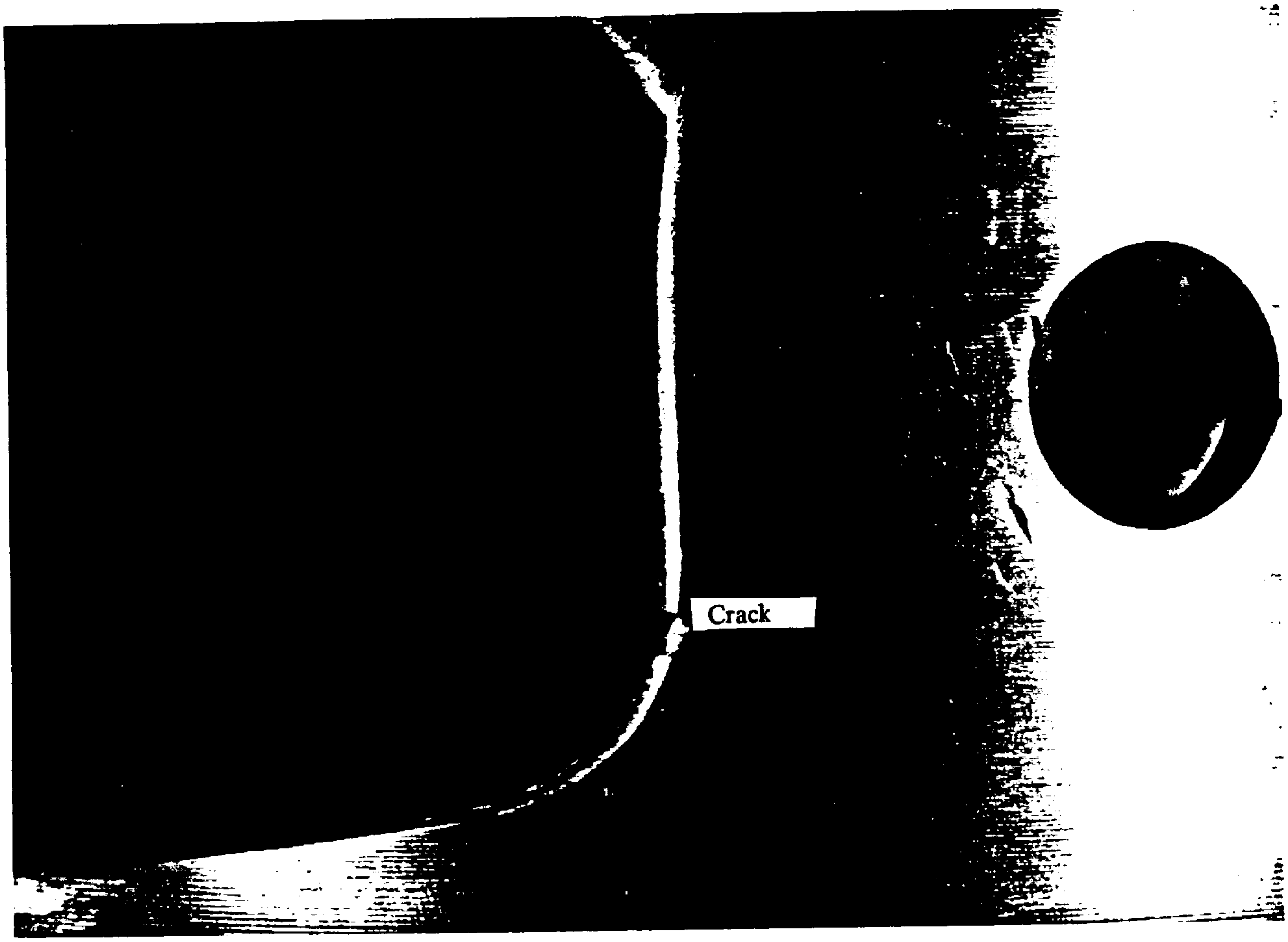


Figure 1. An unprocessed image showing a crack emerging from a machine part.

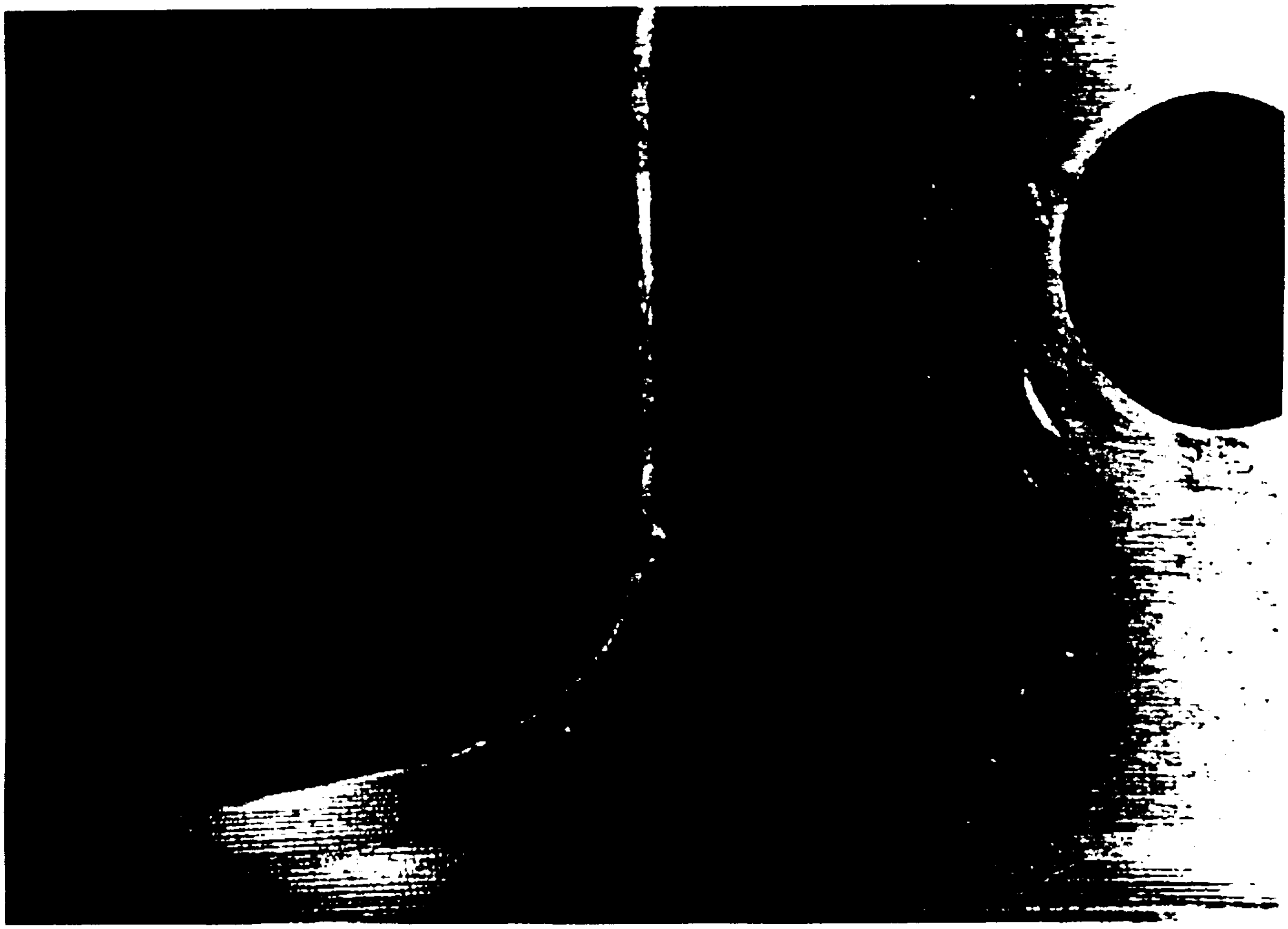


Figure 2. An image of the crack in the workpiece after magnetic particle material has been applied.

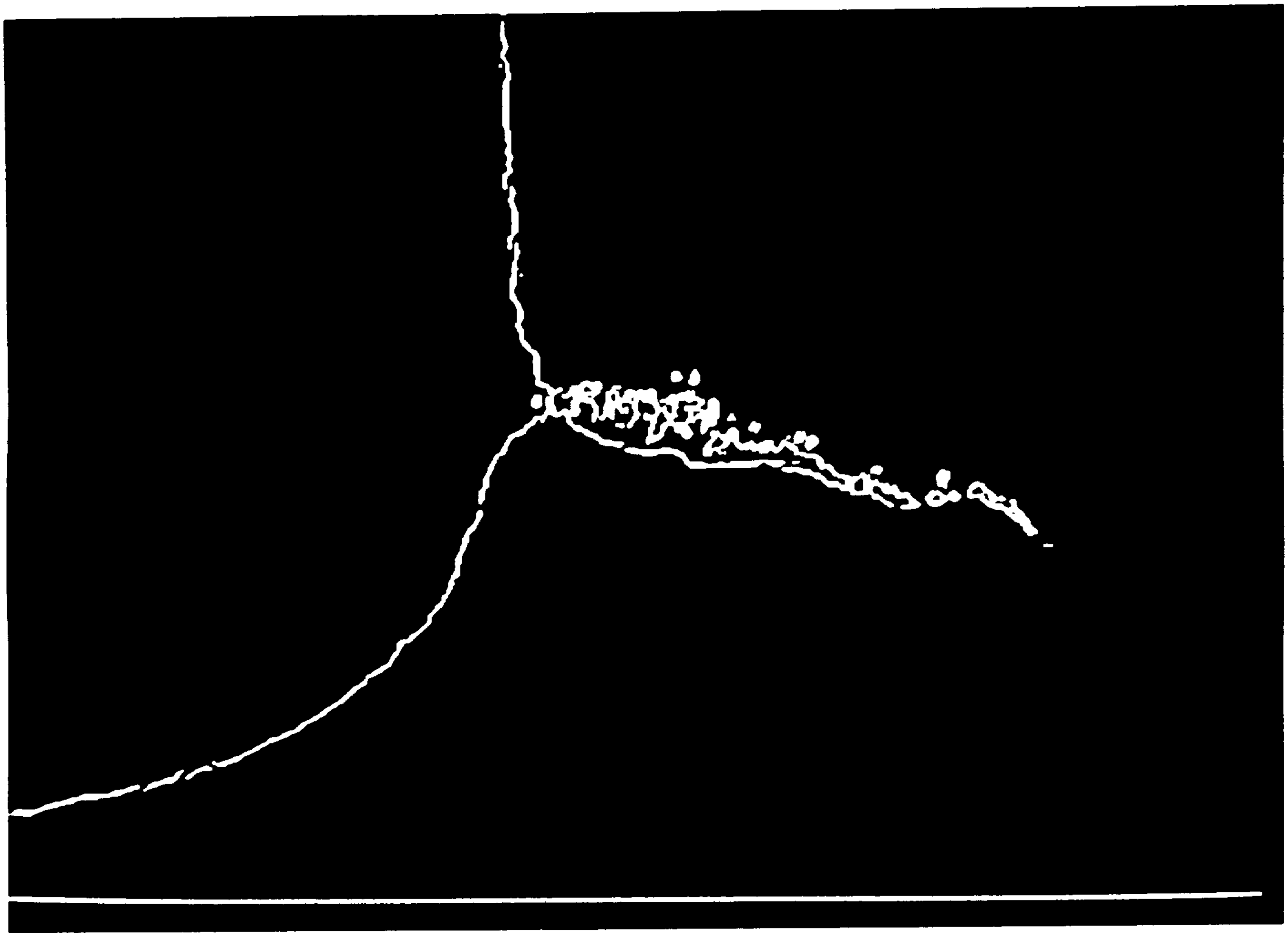


Figure 3. The image of the crack after edge detection algorithm has been applied.

Analysis Using Artificial Neural Networks

Artificial neural networks (ANN's) have gained prominence recently among researchers of various practical problems which have not been solved adequately by traditional methods. As the name implies, these networks are computer models of the processes and mechanisms that constitute biological nerve systems, to the extent that they are understood by researchers.

Interest in artificial neural networks has grown rapidly in the past years as they have been envisioned by many as an alternative to artificial intelligence (AI) as a way to attack problems that "traditional" AI methods have been unable to solve. Most of the growing interest in neural networks has surfaced since the mid-1980's and by now, at the beginning of the 1990's, neural network conferences are arranged by professionals from such diverse areas as science, banking and financing, engineering, and the military. The widespread interest in neural networks stems from their ability to solve complex problems which so far have proved difficult for competing methods.

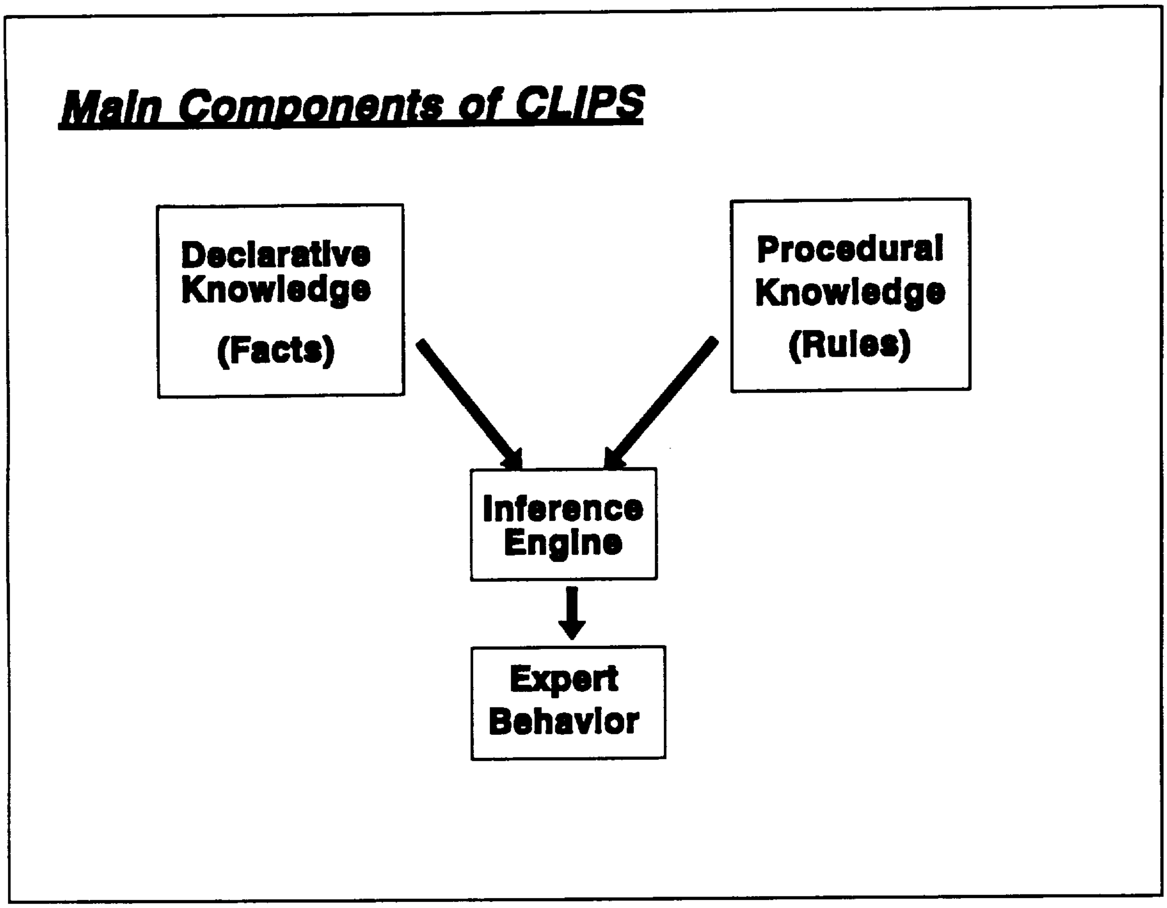
In addition to the mere ability to conquer complex problems, neural networks are attractive in view of their high execution speed and their relatively modest computer hardware requirements. Beyond the variety of personal computer and workstation implementations that are available commercially, several efforts are presently underway to take advantage of the inherent parallelism of neural networks. The calculations taking place within the networks are not generally carried out in a serial fashion as they can be conducted in parallel. Although present computers implement neural networks very efficiently, based on the traditional, serial, Von-Neuman architecture, emerging parallel computers will be ideally suited for neural network implementation.

Neural networks were used in this work to analyze weld profiles and determine the location of certain features, such as the bead center, undercuts, the edges of the bead, etc. This technique has been demonstrated to be particularly effective in analyzing the VPPA weld profiles. Specific examples are given in chapter V, and further discussion on artificial neural network is provided in appendix A of this report.

Rule-Based Expert Systems

The use of rule-based expert systems for weld quality assessment was examined during the course of this work. A number of approaches exist to process the information obtained from the image processing routines and expert systems were studied for this purpose. Among the questions that arise are what constitutes an "acceptable" workpiece or specimen, and to what extent defects can be tolerated.

A number of software tools were surveyed for the purpose of making PASS/FAIL decisions. A relatively logical candidate was the CLIPS (C Language Integrated Product System), which has been developed by NASA at the Johnson Space Center [9], and is available to NASA contractors for work such as this one. This system allows the user to specify a collections of general "rules" and define a collection of "facts" relating to the problems at hand, as shown in figure 4. The CLIPS "inference engine" is used to iteratively compare the rules against the facts, which in turn derives sets of new facts, and thus the system continues until no further deductions can be made.



Main Components of CLIPS

Declarative Knowledge (Facts)

Procedural Knowledge (Rules)

Inference Engine

Expert Behavior

Figure 4. A schematic diagram of the CLIPS rule-based expert system.

A small expert system was constructed to examine the feasibility of determining weld quality based on observed defects. Representative samples of the facts list and the rule base are listed below.

Facts:

```
(deffacts qc-facts
   (defect-types longitudinal-crack crater-crack transverse-crack))
```

Rules:

Rule 1

```
(defrule longitudinal+crater-crack
   (longitudinal-crack) &                              (IF....)
   (crater-crack)
   =>
   (assert (quality = 4)))                             (THEN...)
```

Rule 2

```
(defrule longitudinal-crack
   (longitudinal-crack)
   =>
   (assert (quality = 6)))
```

Rule 3

```
(defrule crater-crack
   (crater-crack)
   =>
   (assert (quality = 8)))
```

Rule 4

```
(defrule transverse-crack
   (transverse-crack)
   =>
   (assert (quality = 5)))
```

Rule 5

```
(defrule porosity
  (porosity)
  =>
  (assert (quality = 6)))
```

The above system and other similar expert systems were evaluated and generally worked as expected. It was found, however, that the main problem associated with the weld evaluation was not in the reasoning process (which the expert system is mostly concerned with) but rather in the image analysis process. The evaluation schemes that were finally selected and demonstrated to yield acceptable results are discussed in chapter V.

CHAPTER V

QUALITY EVALUATION

The task of weld quality evaluation is not trivial, even for the experienced quality inspector. This is particularly true when it comes to specifying in quantitative terms what attributes of the weld affect its quality and to what extent. Different types of discontinuities have been categorized for this purpose, such as cracks, porosity, undercuts, microfissures, etc. The task of quantifying the presence of these discontinuities and relate this to the overall quality of the weld is not easily specified, and the task is even more difficult when it comes to coding them in terms of computer operations.

An overview of the quality evaluation process is outlined in figure 5. The block diagram contains the functions which are carried out in the quality evaluation process. Examples of particular types of each function are shown on the left. All or none of these at each step may be used in any particular evaluation. The workpiece to be inspected may optionally be treated by a variety of standard pre-inspection methods, such as magnetic particle treatments or dye penetrant application. The piece is appropriately illuminated with white light, colored light, structured light, or laser illumination, and an image is captured and digitized. Various image enhancement procedures are available to extract features from the image, such as enhance cracks, porosities or other artifacts. The image processing techniques applied to the digitized picture include integration, thresholding and other standard image processing methods. Finally, an evaluation of the inspected weld is carried out, resulting in a quality assessment of the inspected workpiece.

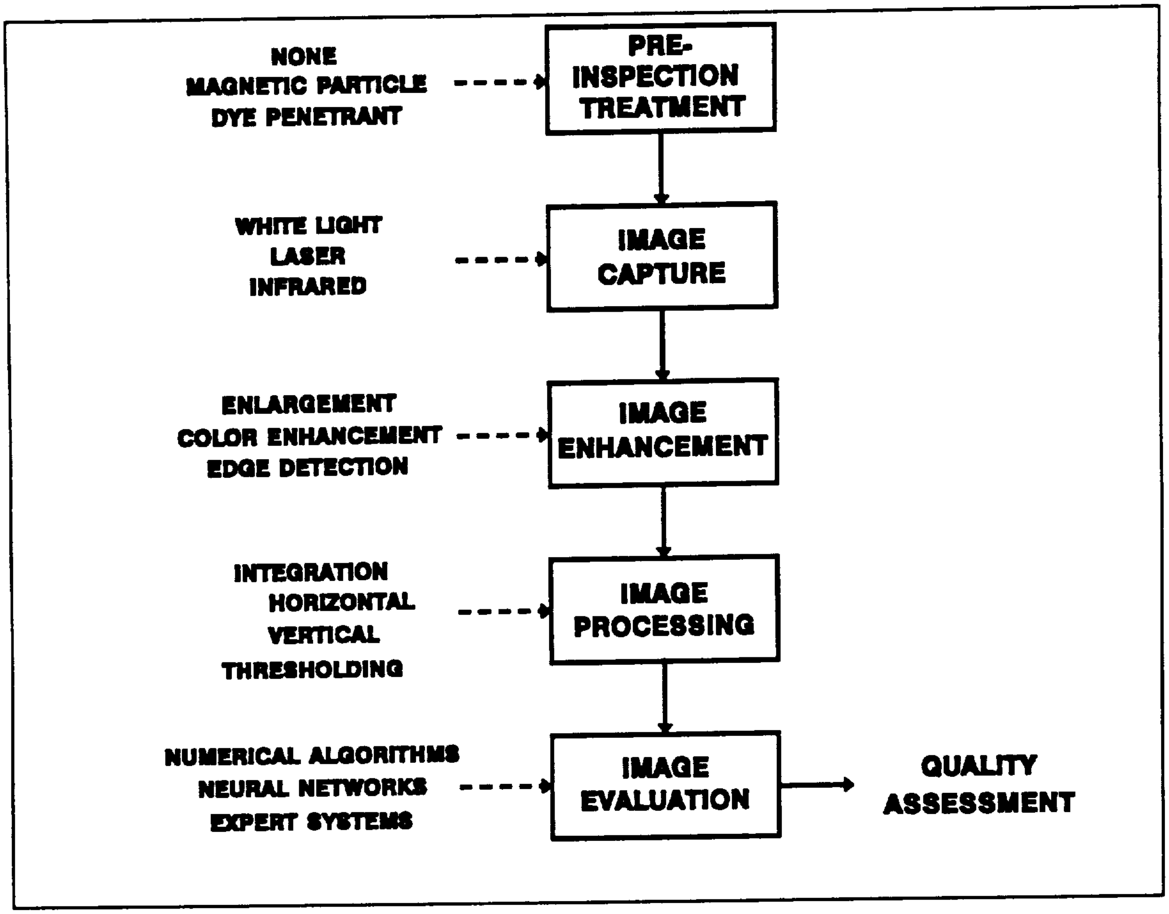


Figure 5. An overview of the quality evaluation process.

The approach taken for this work was to demonstrate the use of computer analysis to evaluate welds based on global characteristics of their appearance. Indicators of weld quality include the general regularity of the weld bead, the relative spacing and regularity of the solidified pool ripples along the bead, and the symmetry of the weld. Other approaches may be taken to the issue of quality control for welding, but the ones used proved to produce consistent and reliable indicators of weld quality when compared against objective human quality judgments.

Due to the different characteristics of various welding processes and the difference in weld appearance, it was found necessary to apply different evaluation strategy to each process. This was done for three common welding processes. The Gas Metal Arc Welding (GMAW) process is a very common welding process and widely used in various commercial welding applications. One of its advantages is the high metal deposition rate, which makes it attractive for high-quantity applications. A continuously-fed wire serves both as the electrode maintaining the arc and as a filler metal which melts into the welded joint. Inert gas flow surrounds the arc and the molten pool. The Gas Tungsten Arc Welding (GTAW) process is widely used for precision work. Unlike the GMAW process, GTAW uses a non-consumable tungsten electrode and filler wire is fed separately into the joint. Finally, the Variable Polarity Plasma Arc Welding (VPPAW) process is of some interest here, particularly because it is increasingly used by NASA and commercial users. It is similar to the GTAW process with two notable differences. First, two sources of inert gas are used: one serves to form the plasma arc, while the other serves primarily for shielding the arc and the pool from the atmosphere. Secondly, the arc current polarity is periodically switched between the nominal electrode-negative mode and brief electrode-positive intervals which serve to cleanse the oxide layer that forms on metals such as aluminum.

Because of the substantial differences between these welding processes, the evaluation strategies will be outlined separately in the following sections.

Quality Evaluation for the Gas Tungsten Arc Welding Process

A good GTAW weld is typically characterized by a regular weld bead surface, as illustrated in figure 6. The surface is free from any indications of cracking, porosities, undercuts or other discontinuities. Furthermore, a regular pool ripple pattern forms along the weld bead. This pattern tends to get irregular and distorted if the welding speed is out of the range that is otherwise suitable for a weld pool of the given size. For comparison, figure 7 shows a typical low quality weld, which would hardly pass any reasonable inspection process.

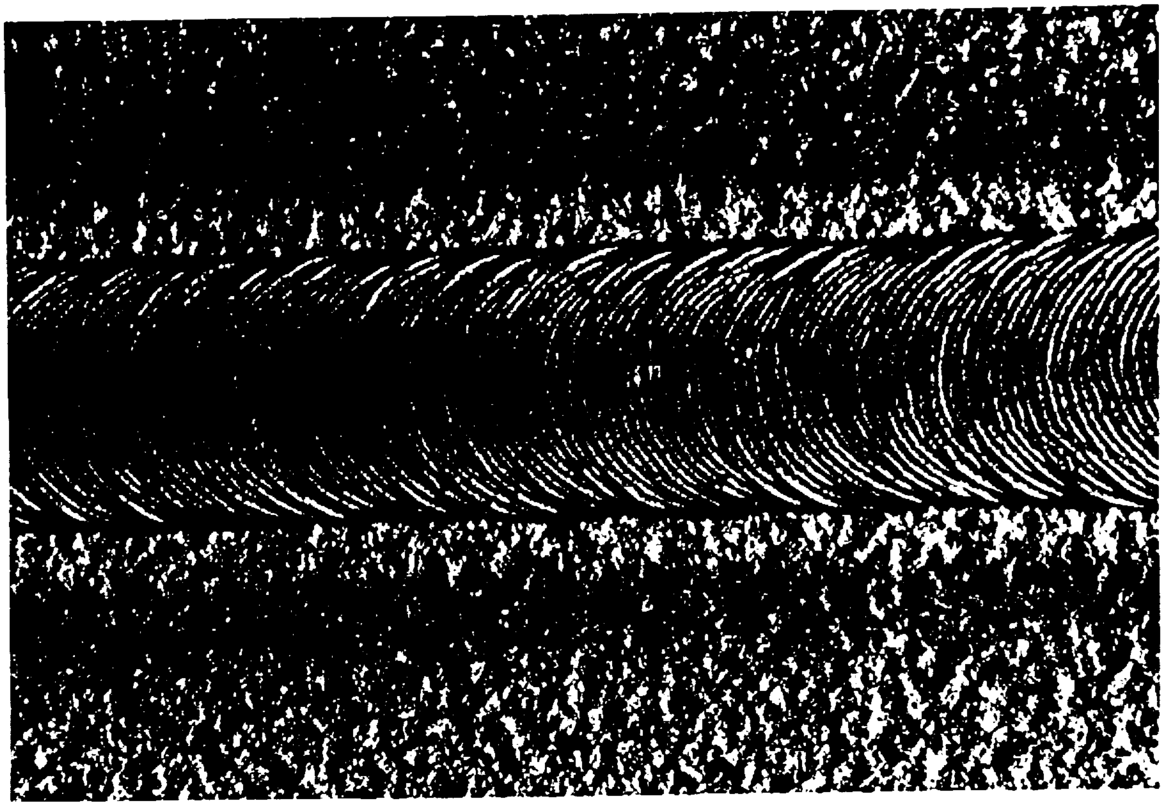


Figure 6. A photograph of a good quality GTA weld bead.

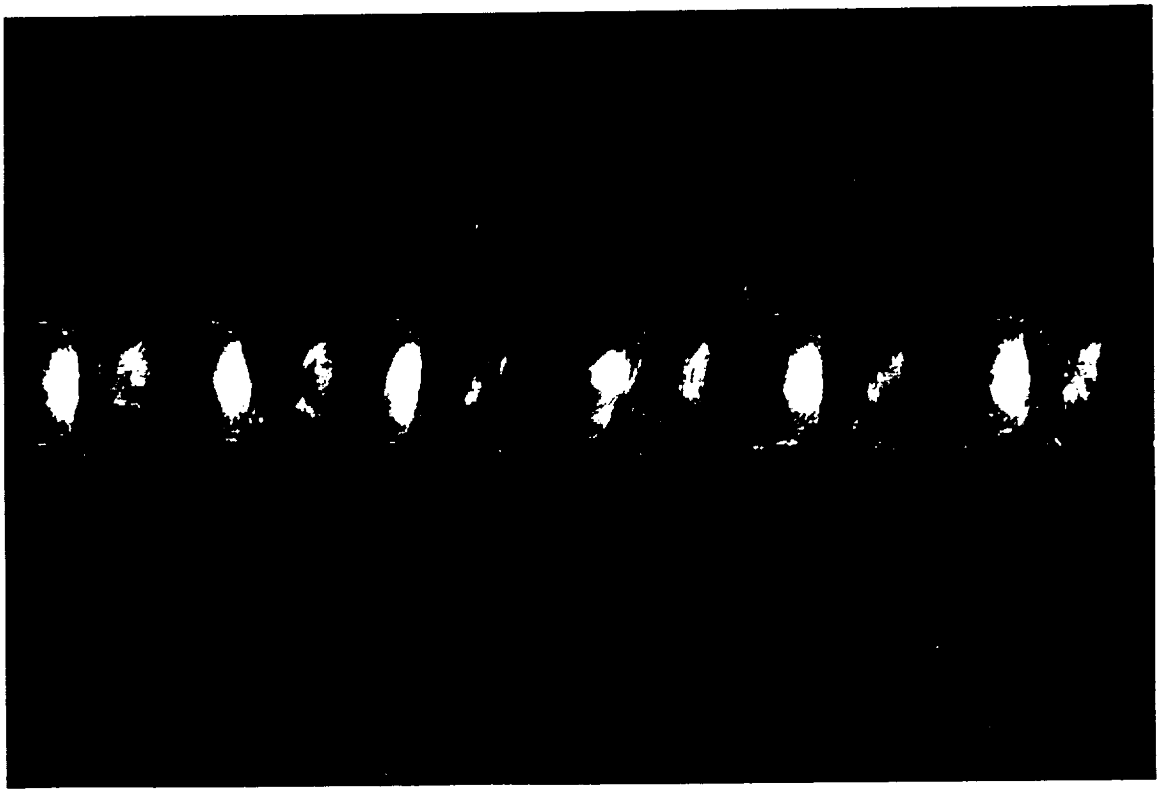


Figure 7. A photograph of a low quality GTA weld bead.

As discussed earlier, the ripple pattern serves an important role in the approach taken here to weld quality evaluation. Two primary criteria were selected for evaluation of the ripple pattern. First, increased regularity generally indicates increasing quality. Skipped ripples or clustering of ripple formation are good indicator of poor welding. A second, and somewhat independent, measure of ripple pattern quality is the average spacing between the ripples. Under certain circumstances relatively regular wide spacing may be apparent, caused by too high travel speed and thus "regular" skips in the ripple pattern. In a summary, a high quality GTA welds may be characterized by (i) regular ripple spacing and (ii) appropriate ripple/bead width ratio.

The ripple pattern can be analyzed through the "integrate" menu selection in the image processing software developed for this work. This operation consists essentially of two tasks. First, the bead width has to be determined from the inspected image, and then the spacing of the ripples along the weld bead must be determined. Generally, the evaluation technique normalizes the ripple spacing to the bead width. By doing this, the focal length of the camera, and the working distance between the camera and the workpiece do not have to considered in analyzing the ripple spacing. The bead width may be estimated with the horizontal integration procedure of the image analysis software. This procedure is graphically illustrated in figure 8. The horizontal dimension of the sampled image measures X_N-1 pixels and the vertical dimension is Y_N-1 pixels. For the GTAW process the window of concern extends between the bottom and the top of the image, and its width may be varied by the user. Integration is carried out along the horizontal (x-axis) direction of the image. The result is a pattern that represents the weld profile as seen along the weld axis, as shown in the figure.

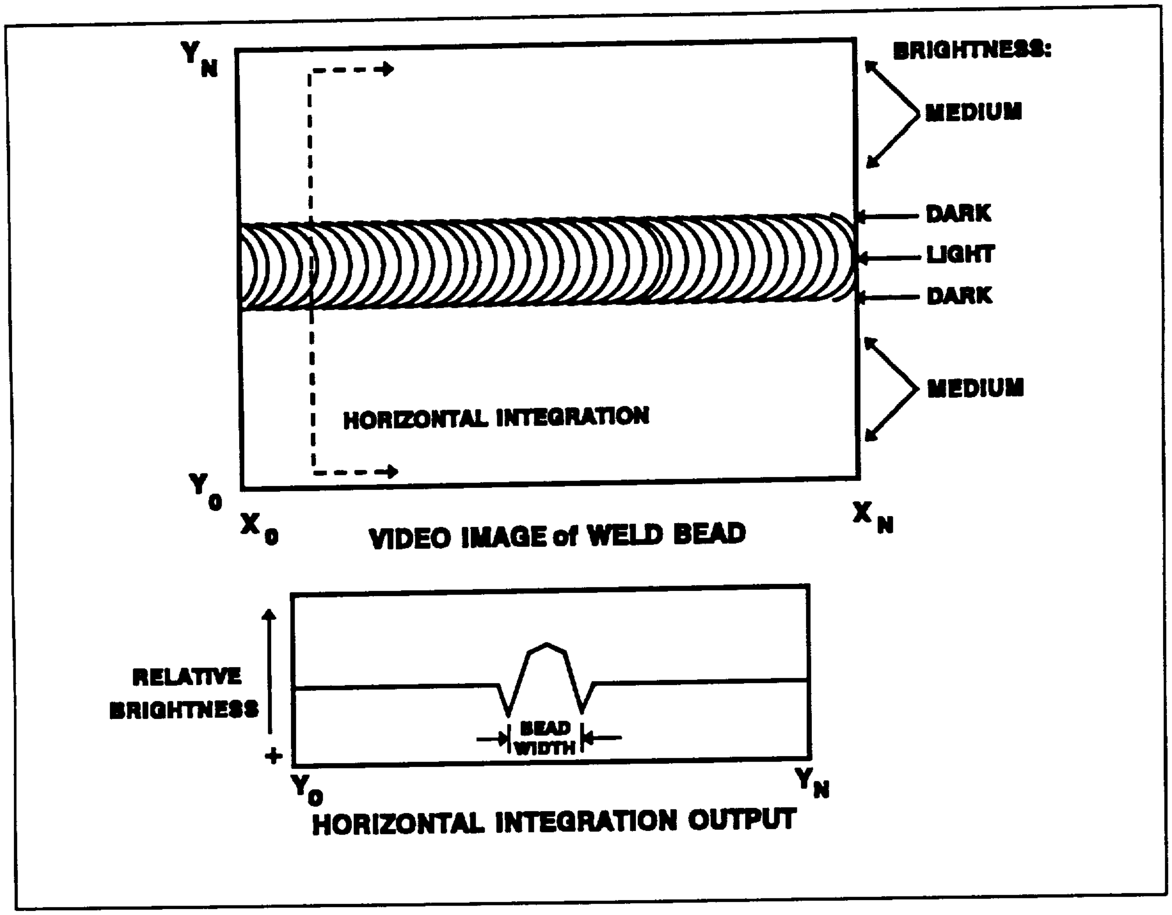


Figure 8. The horizontal integration procedure illustrated. This technique is used in analysis of GMA weld quality.

Actual image data obtained through the integration process is shown in figure 9. The shape of the weld profile is clearly recognized, with a surface peak at the center and the bead edges on both sides. Although the general pattern is clearly recognized by the human eye, a computer recognition of the bead features is not straightforward. Using filtering, however, the signal can be improved. Figure 10 shows the same pattern, filtered by 21-pixel running averaging, which presents a better signal for computer processing. The bead profile is used here to estimate the bead width to establish a reference for the bead ripple spacing.

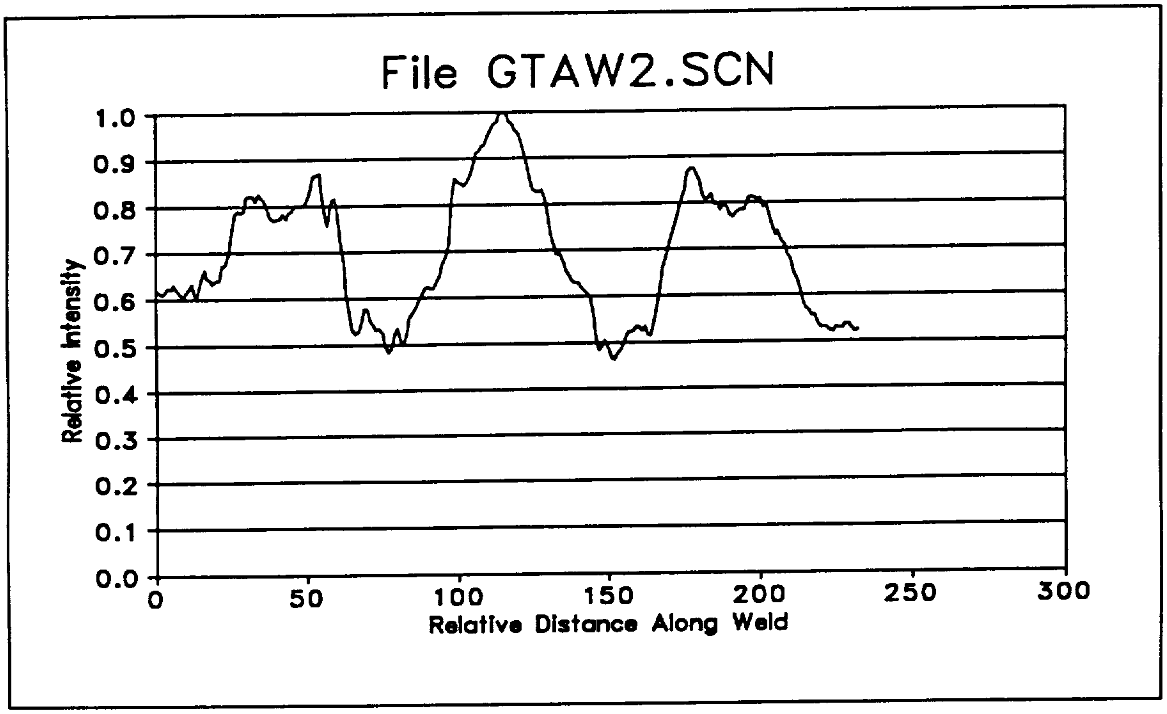


Figure 9. Image scanning data representing a GTA weld profile.

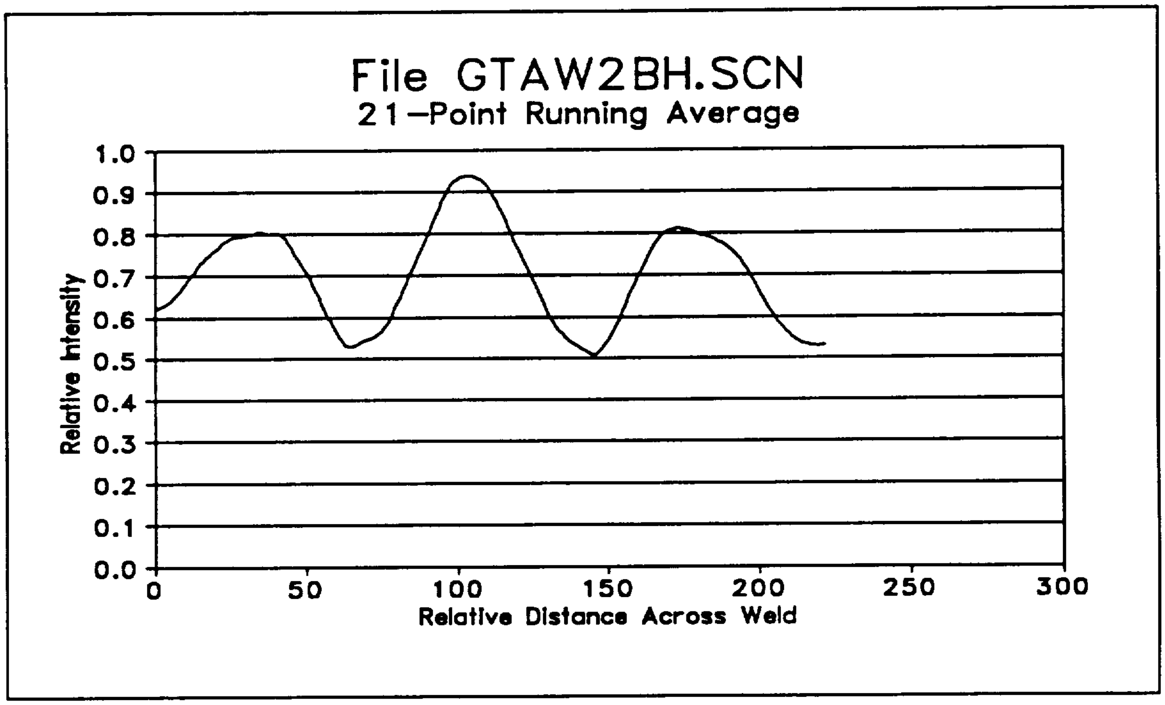


Figure 10. Image scanning data representing a GTA weld profile, filtered to aid further computer analysis.

A 16-line high window was used for the horizontal integration procedure, and the brightness intensities of the 16 pixels of each horizontal position were integrated to obtained a somewhat smoothed image of the ripple pattern along the weld bead. The vertical integration technique is graphically illustrated in figure 11. The horizontal dimension of the sampled image measures $X_N\text{-}1$ pixels and the vertical dimension is $Y_N\text{-}1$ pixels. For the GTAW process the window of integration is generally placed along the centerline of the image. The ripple pattern is obtained from the specified window, as shown at the bottom of the figure.

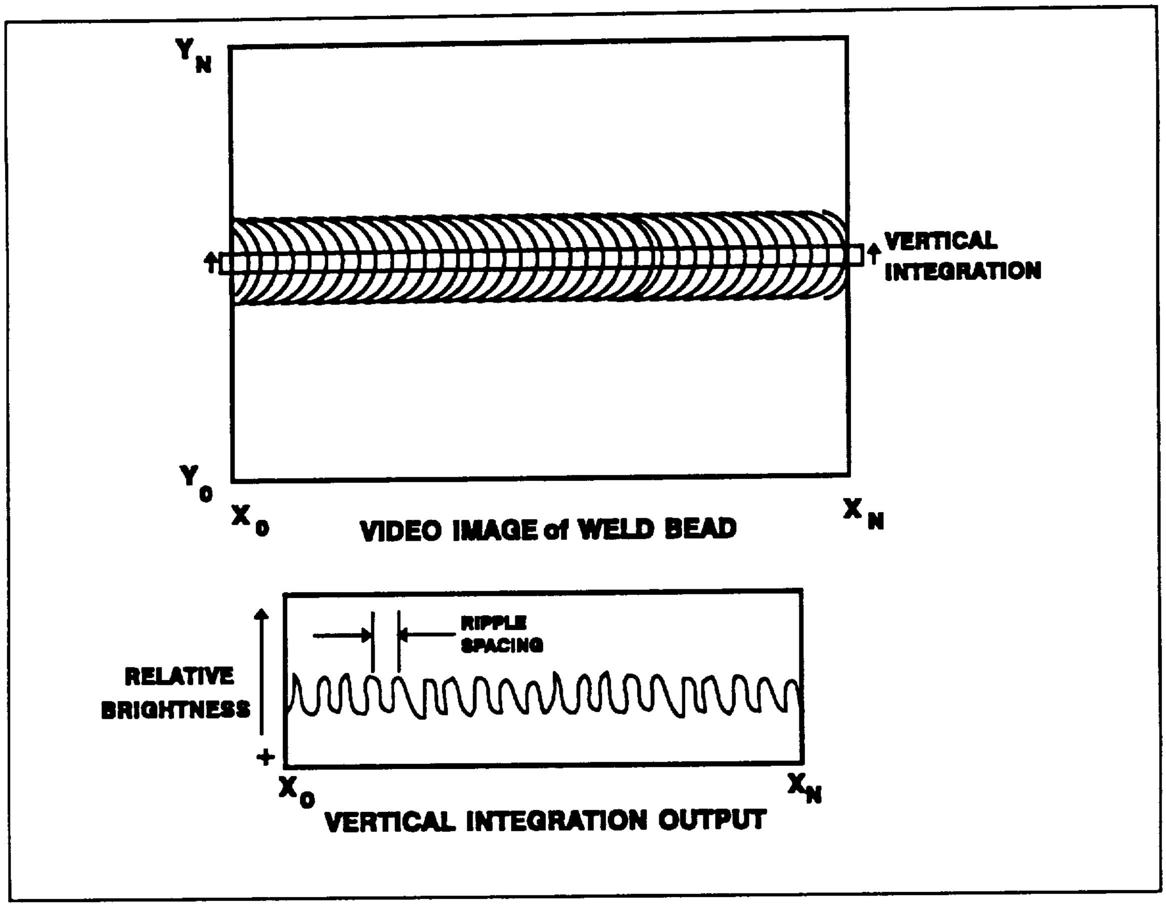


Figure 11. The vertical integration procedure illustrated. This technique is used in analysis of GTA weld quality.

A variety of methods were investigated to yield information regarding the weld ripple spacing characteristics. The Fourier transform is a natural candidate for this task. A regular ripple pattern with consistent spacing should yield a Fourier spectrum with a single, distinct peak at the frequency corresponding to the ripple spacing. Furthermore, a narrow peak should indicate small variations in ripple spacing, while a broader peak would result from larger variations in the pattern spacing. The Fourier transform was taken of several ripple pattern and examined. Typically, the Fourier spectra turned out to be very irregular, as figure 12 illustrates. This spectrum was obtained from a weld which had been evaluated to be of a relatively good quality. Despite a dominating spectrum peak, corresponding to the dominating ripple frequency, the irregularities are severe enough to produce additional peaks of significant amplitude. Typically, the power spectrum was much less clear for welds of less quality, and frequently no spectrum peak could be determined to be the dominant one.

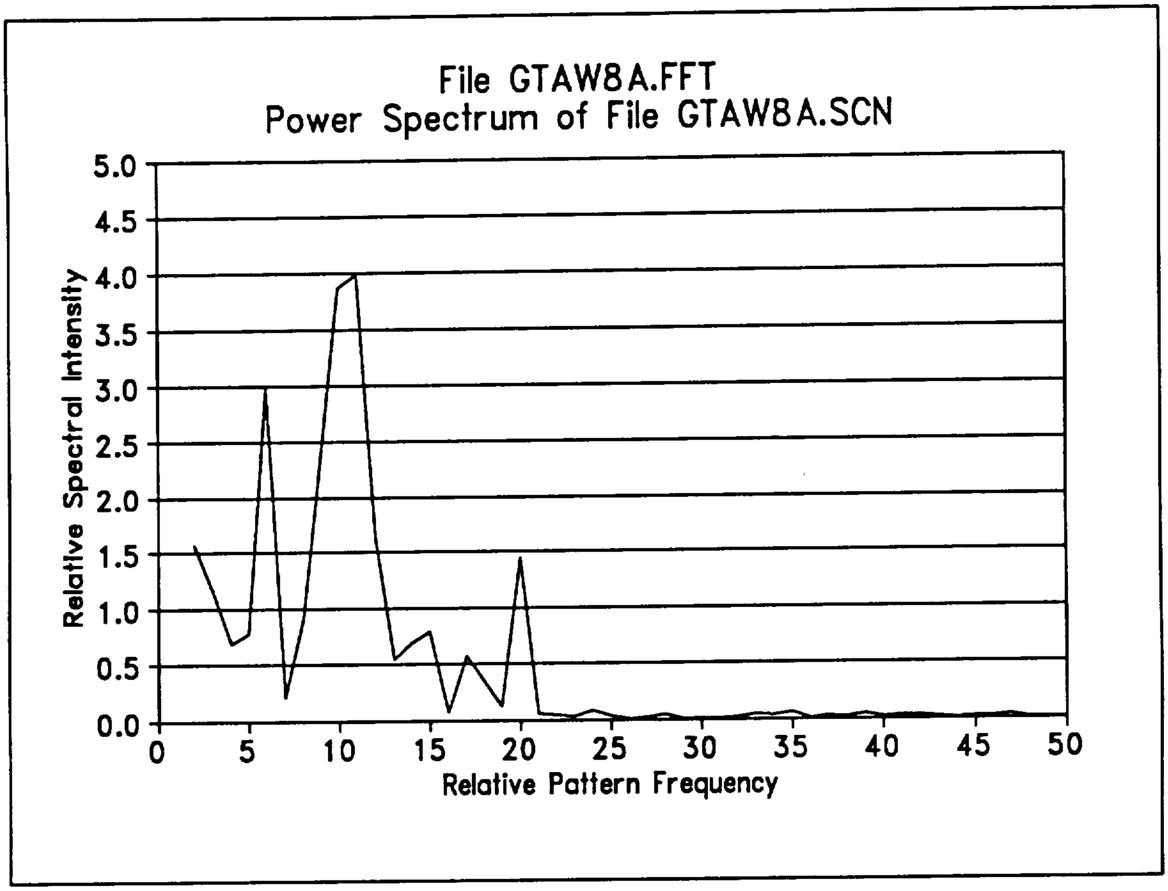


Figure 12. The vertical integration procedure illustrated. This technique is used in analysis of GTA weld quality.

Various smoothing techniques were tried to clarify the Fourier spectrum, in order to enhance further the primary frequency peak. None were found to work satisfactorily over a considerable range of weld qualities. In addition to Fourier analysis, autocorrelation techniques and other related methods were investigated, but none yielded reliable, consistent results.

Following the investigations outlined above, a new technique was adopted for weld evaluation. The selected approach to computerized GTA weld quality evaluation is summarized in figure 13. The video image of the bead surface is sampled, digitized, and stored in computer memory. Two aspects of the weld bead are quantified: the bead width and the relative density (spacing/bead width, or frequency) of the ripple pattern along the weld. These quantities are obtained by analysis of windows from the image that have been integrated along the horizontal and vertical dimensions, respectively. The ripple pattern frequency and the variance of this frequency are used to assess the overall quality of the GTA weld bead.

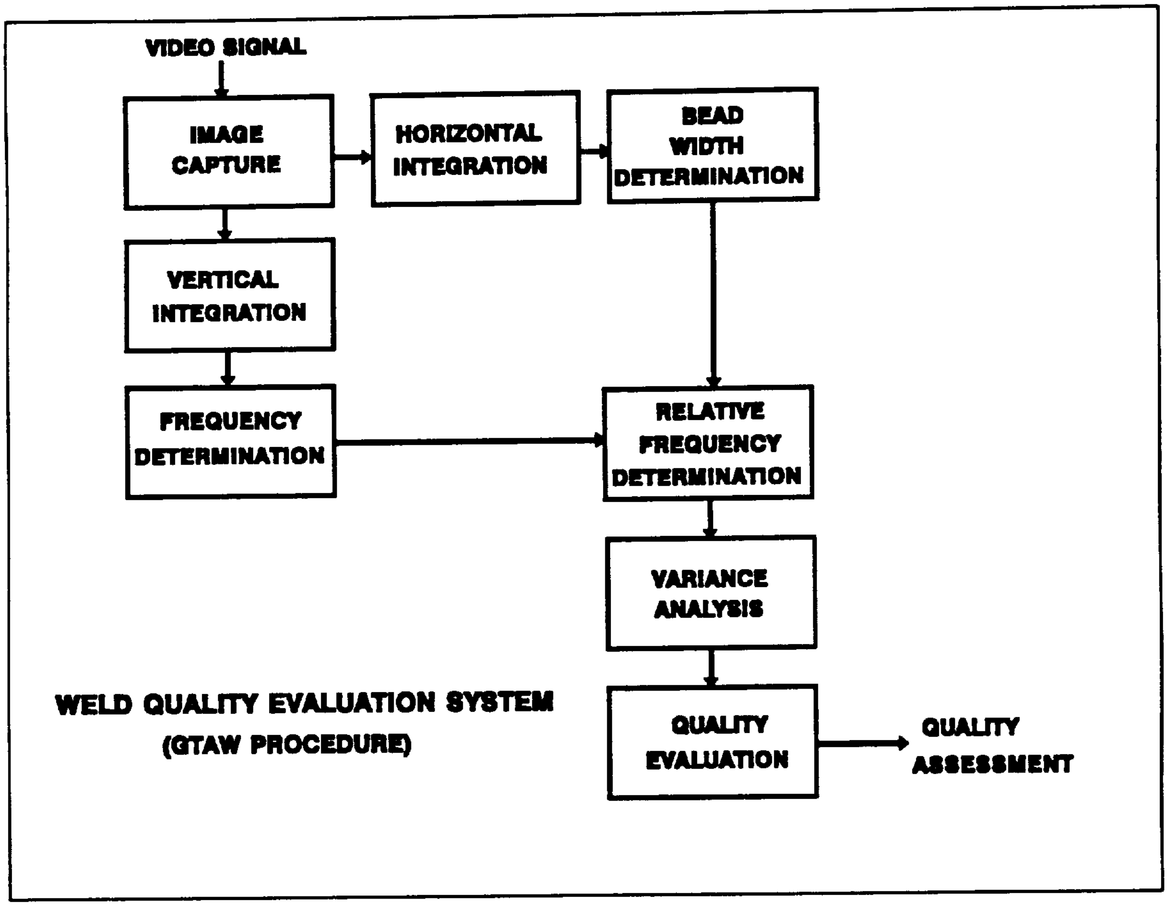


Figure 13. An overview of the process developed for evaluating GTA weld quality.

To demonstrate further the technique used for quality evaluation, four GTA welds were arbitrarily selected and evaluated on a scale from 0 to 10 by a human expert. The GTAW quality evaluation process was based upon observation of three major criteria. The first was regularity of spacing of the ripple pattern on the surface of the weld. Uniformity in this aspect of the weld's appearance was considered essential to assignment of a high quality rating. However, regularity alone was not sufficient to achieve a high quality rating. The relative frequency of the ripple spacing compared to the bead width was also a significant factor in the assessment process. A low relative frequency (approximately 1/1) was given a lower rating than a high relative frequency (approximately 10/1). The third factor in the rating process was the presence or absence of faults such as cracks, porosity, abrupt changes in lateral or vertical cross section, or other observable defects.

For the GTAW test samples, a graphically produced "ideal" weld, with perfectly uniform ripple spacing, with excellent relative frequency, and with perfectly uniform geometry, was given the score of ten. A real weld, with excellent uniformity, excellent relative frequency, and no significant geometrical anomalies, was given a rating of nine. This weld was shown in figure 6. A weld with good uniformity, but with very poor relative frequency, and severe variations in geometry, was given the rating of two. This weld was shown in figure 7. Between these extremes, other welds were inspected and given subjective ratings based upon the presence or absence of the factors noted above. The welds will be examined further as the general technique is illustrated.

Once the ripple patterns had been digitized they were filtered for noise reduction. The filtering method was a 5-pixel running average. In other words, the intensity of pixel *k* was replaced by the average of the intensities from pixels *k-2* through *k+2*. The original ripple patterns and the filtered versions are shown in figures 14 through 21. The human-assigned weld quality is indicated in the name of the file storing each ripple pattern. For example, file GTAW2.SCN stores scanned ripple data from a GTA weld that was evaluated of quality 2. A number of welds were originally examined for this purpose and whenever more than one weld was assigned the same quality score by the human expert, successive welds were designated by letter suffixes, such as 2A, 2B, etc. Thus, the welds illustrated here, GTAW2, GTAW4B, GTAW6 and GTAW9B, represent a broad range of weld quality levels. Weld quality level 10 was simulated by use of a graphically produced weld image with perfect uniformity of width and ripple spacing.

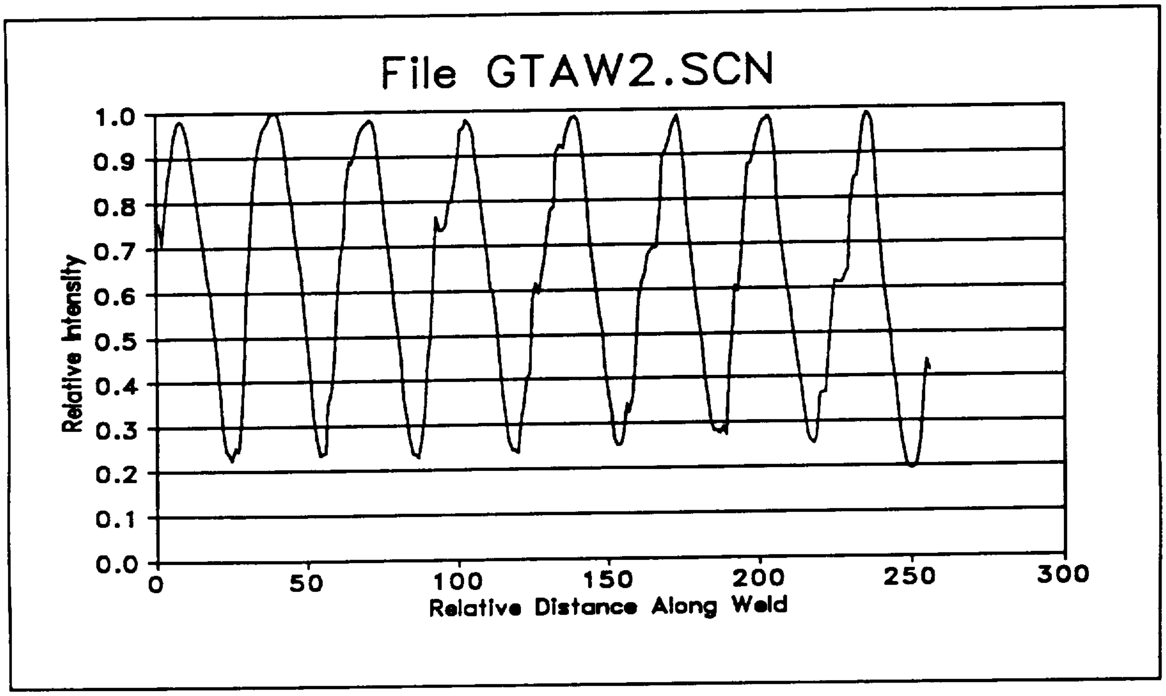


Figure 14. Scanned ripple pattern of weld GTAW2 (human quality evaluation = 2).

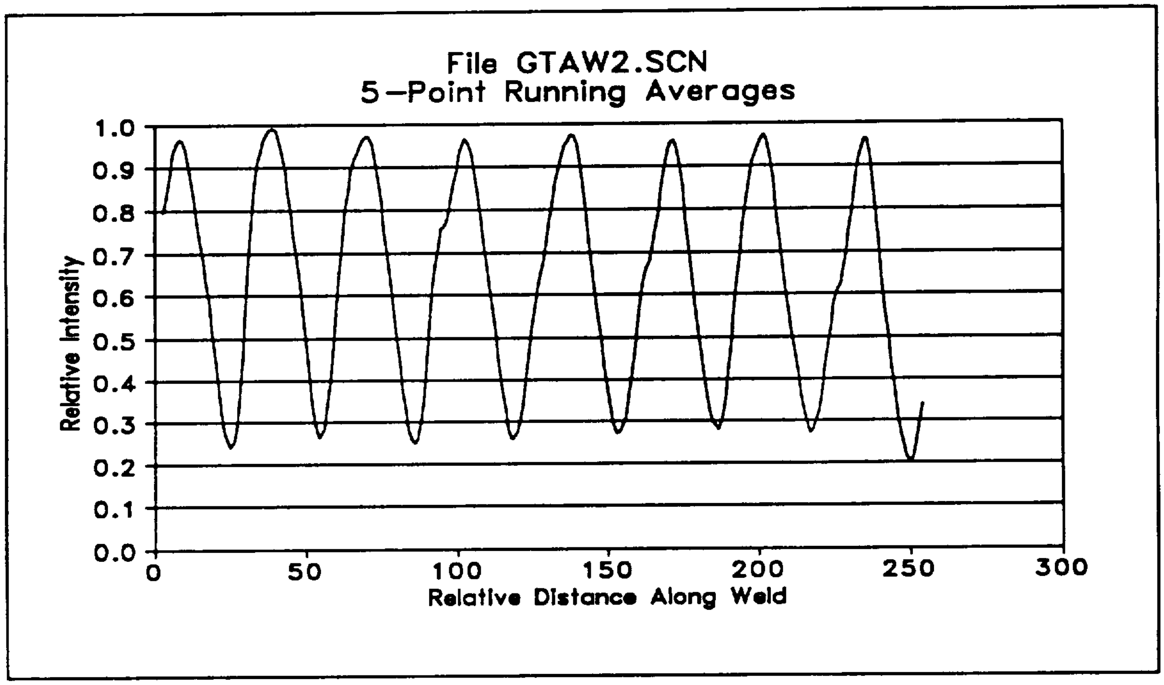


Figure 15. Filtered ripple pattern of weld GTAW2 (human quality evaluation = 2).

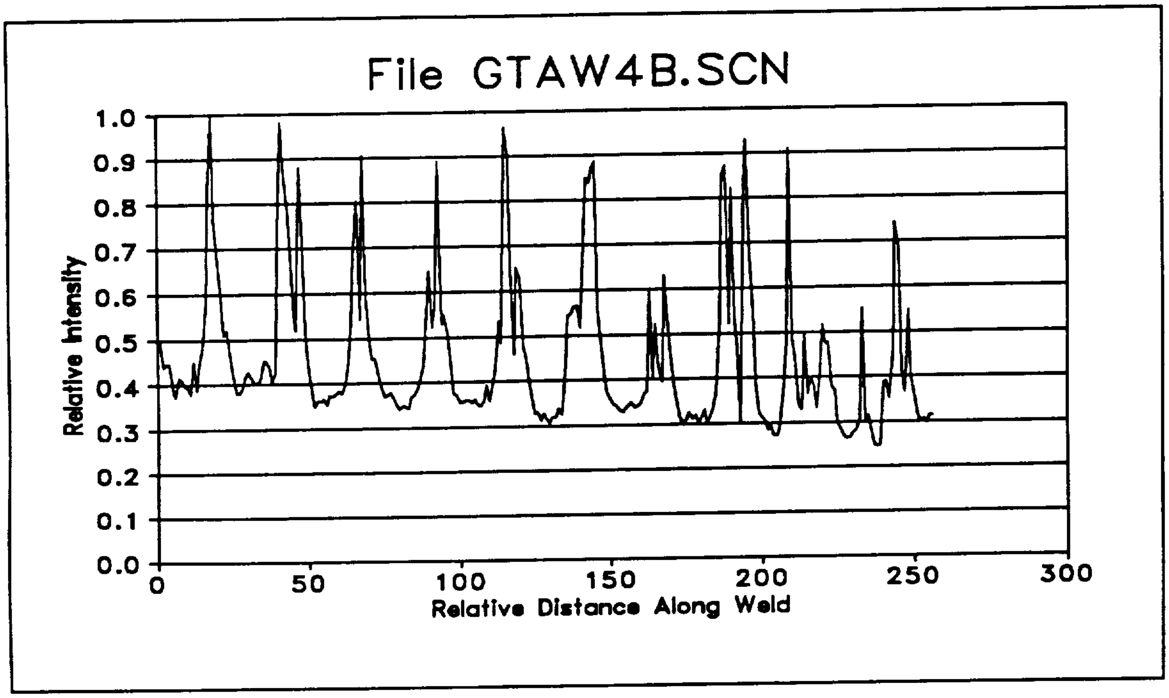


Figure 16. Scanned ripple pattern of weld GTAW4B (human quality evaluation = 4).

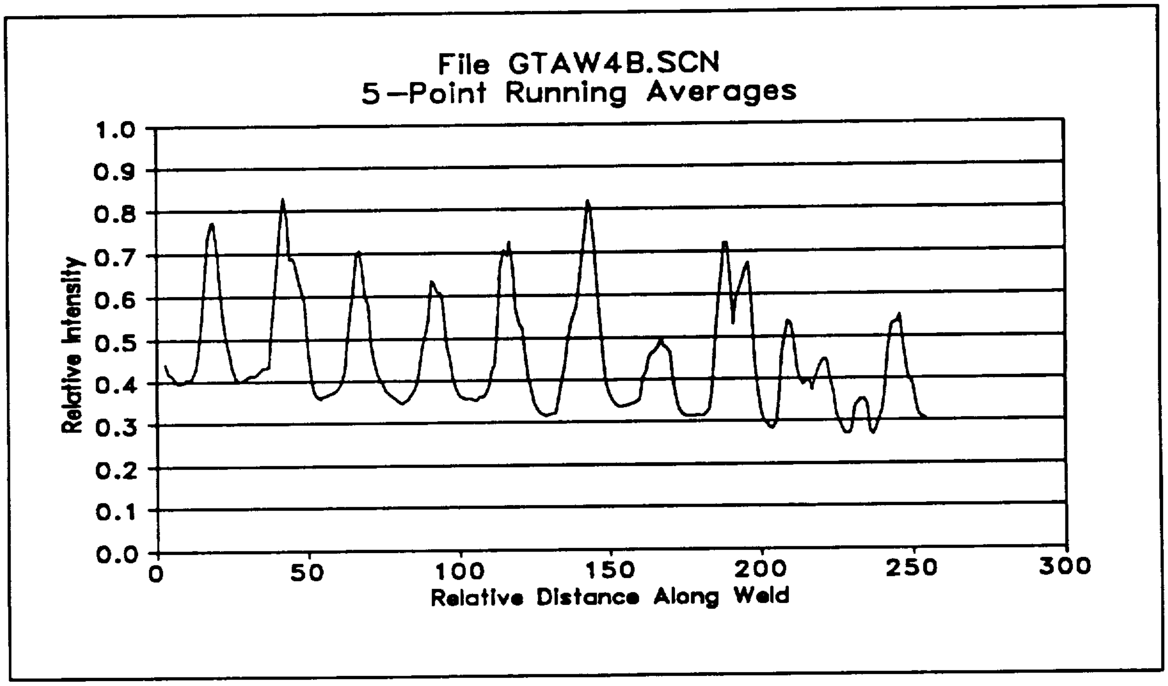


Figure 17. Filtered ripple pattern of weld GTAW4B (human quality evaluation = 4).

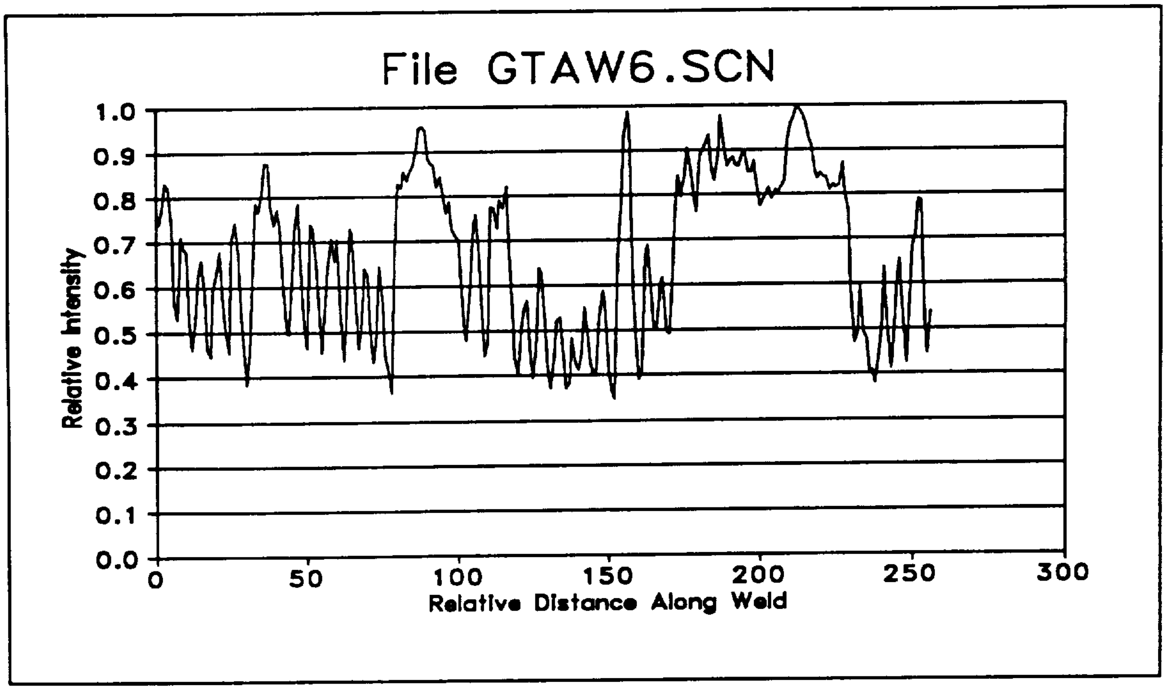


Figure 18. Scanned ripple pattern of weld GTAW6 (human quality evaluation = 6).

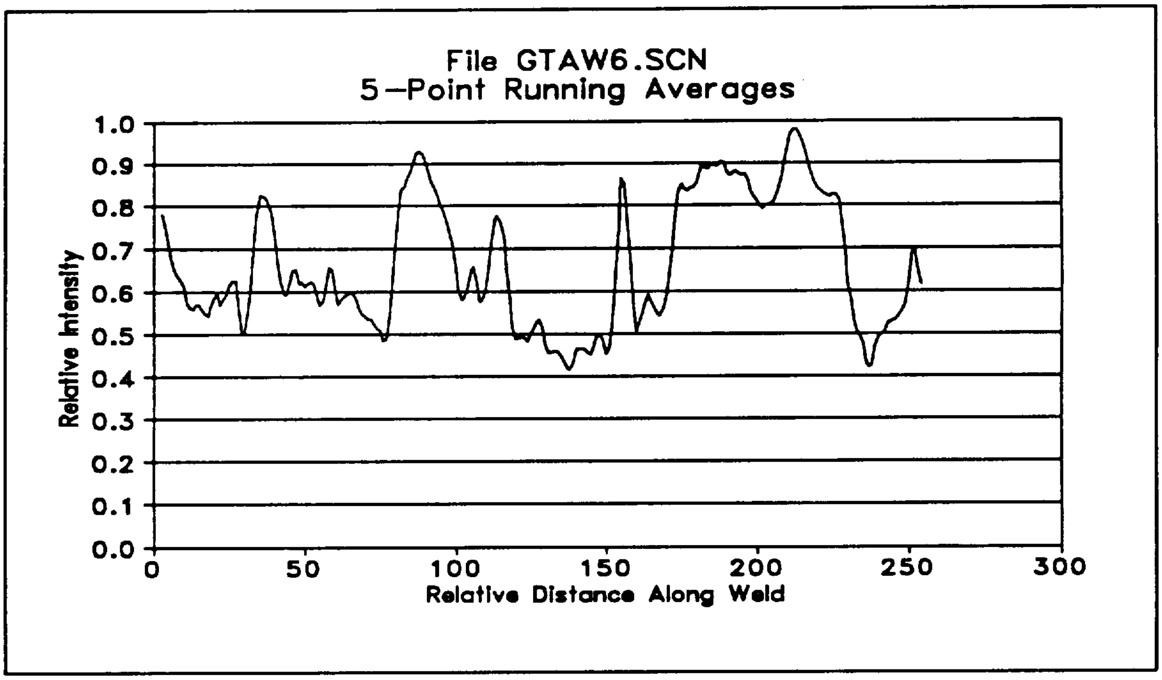


Figure 19. Filtered ripple pattern of weld GTAW6 (human quality evaluation = 6).

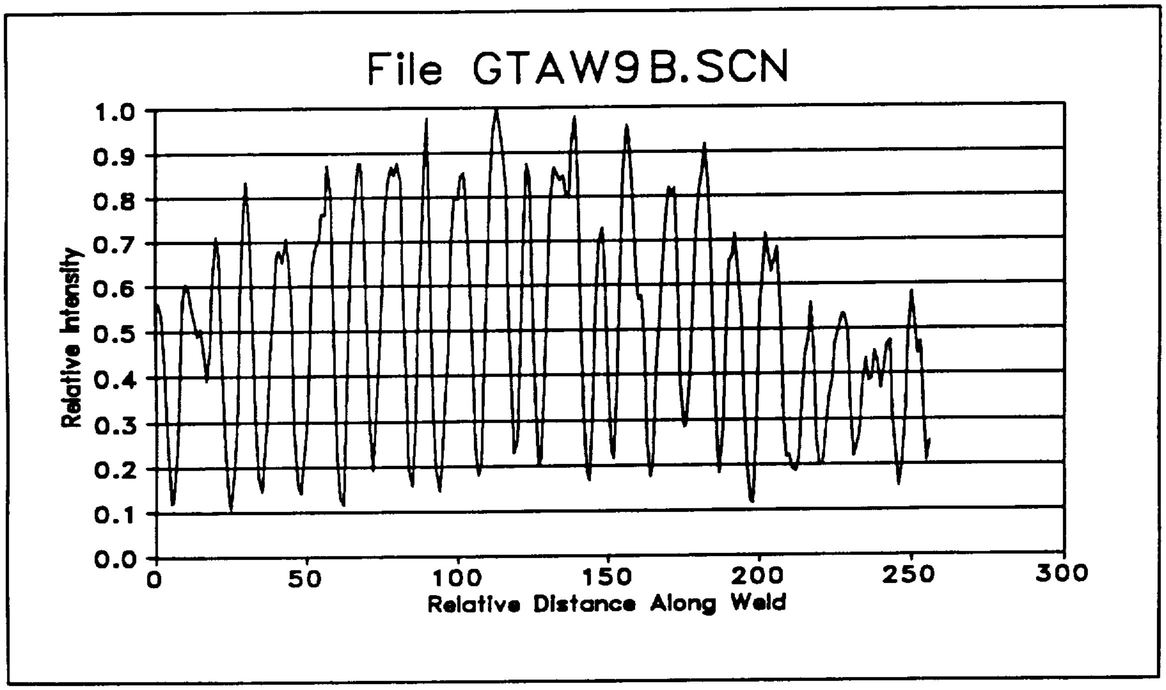


Figure 20. Scanned ripple pattern of weld GTAW9B (human quality evaluation = 9).

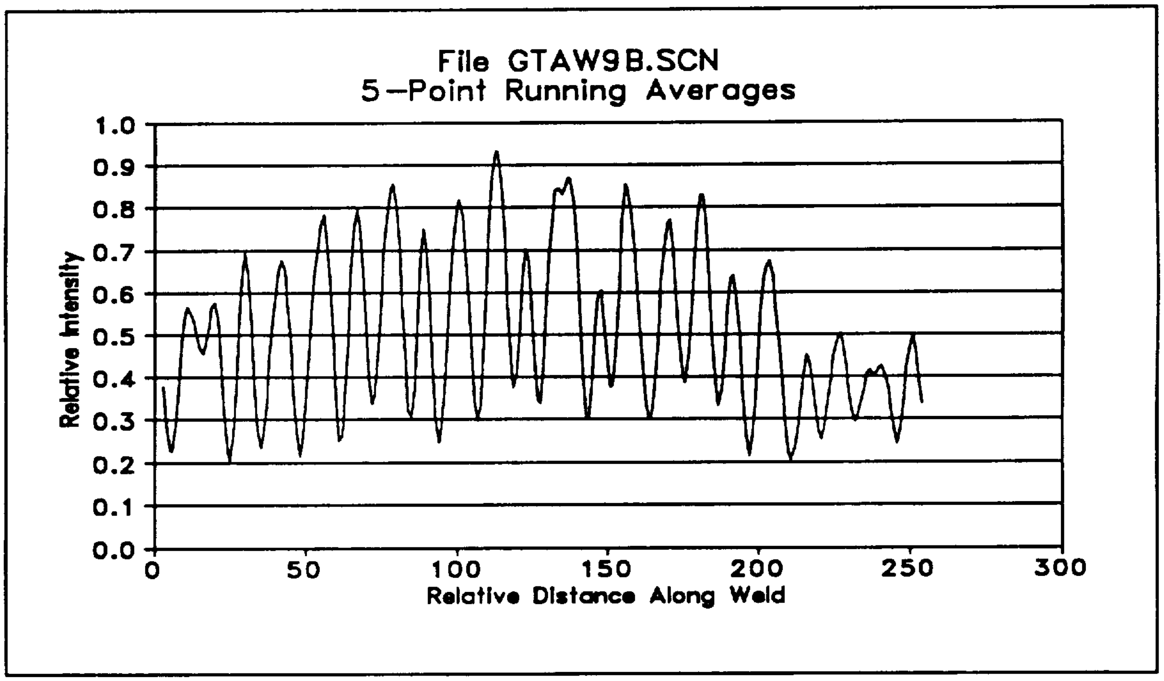


Figure 21. Filtered ripple pattern of weld GTAW9B (human quality evaluation = 9).

Once the ripple patterns had been obtained through the image digitization system and filtered accordingly, a simple algorithm was used to determine the pattern frequency and regularity. First, the spacing between adjacent ripples could be determined by measuring the distance between adjacent ripple peaks in the filtered graph. A pixel was determined to be at an intensity peak if both the pixel immediately preceding it and the pixel immediately following it had lower intensity values than the examined pixel. Using this classification, the ripple spacings were determined. To eliminate the effects of differing weld sizes, the ripple spacing values, originally determined as number of pixels, were normalized to the width of the weld (in number of pixels). Referring to the two quality criteria stated earlier, a good weld is indicated by relatively dense ripple pattern (small spacing/bead width ratio) and pattern regularity (small standard deviation of the measured ripple spacings). Using these two criteria, the rule used here for quality evaluation was:

$$Q = 1 / [(\text{average relative spacing}) + (\text{standard deviation of spacing})] \tag{9}$$

where Q is a positive numerical indicator of weld quality and increases as the quality of the examined weld improves. The average spacing between adjacent weld ripples is determined as described above and the standard deviation is found similarly.

The quality values of the examined welds were calculated and graphed against the quality value assigned by the human inspector. This is shown graphically in figure 22, where the human-assigned quality values are assigned to the horizontal axis and the vertical axis represents the calculated quality indicators. It should be noted that exact numerical correspondence between the human indicator and the calculated one is not sought. Of main concern is that the graph is monotonic, which ensures that the human and the computer agree on comparison between individual welds. In other words, if the human evaluates weld B to be of better quality than weld A, the computing algorithm should agree with this judgement.

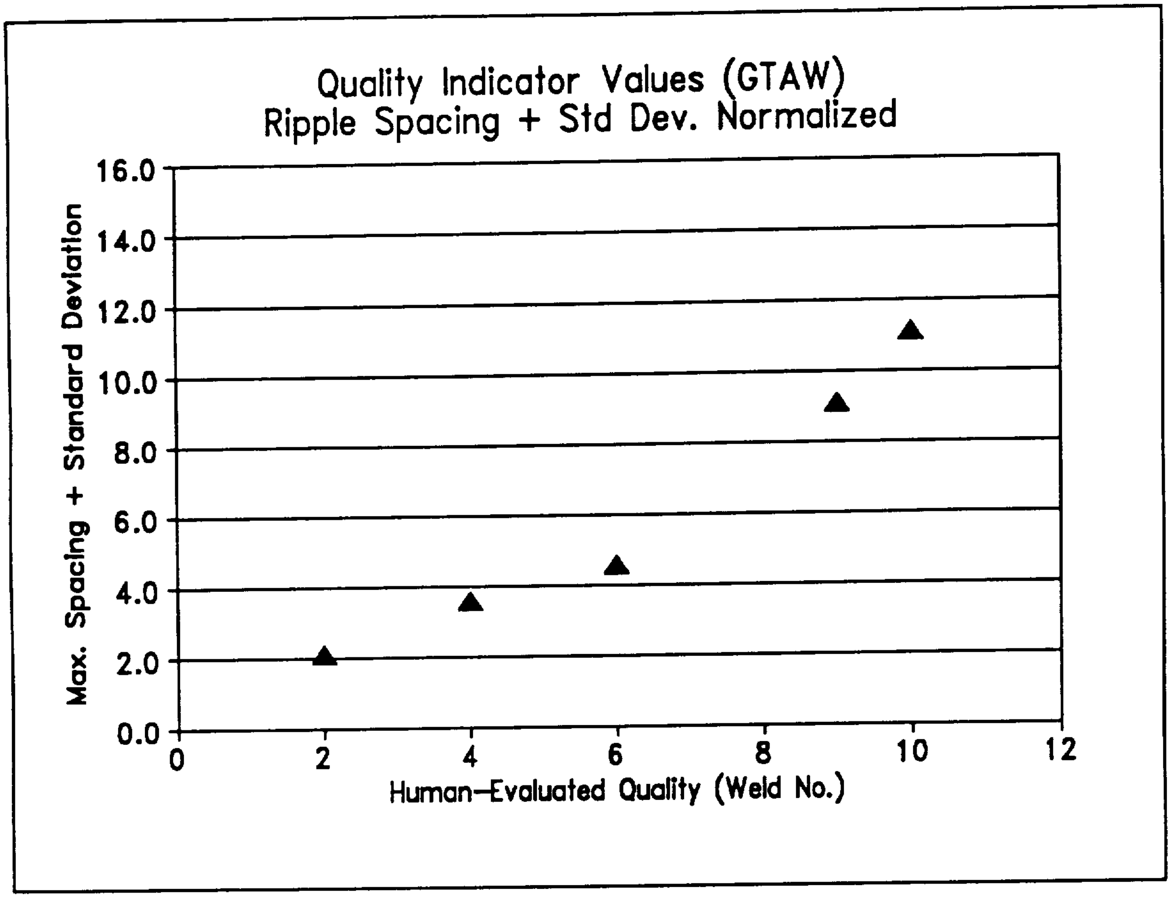


Figure 22. Comparison of the calculated quality indicator and the corresponding human quality evaluations.

As seen in figure 22 the algorithm developed for evaluating GTA welds is consistent and generally agrees with the quality evaluations of the human expert. The technique developed for this work resulted from the desire to develop algorithms which strongly parallel the human reasoning process in evaluating the ripple patterns on the weld surface. Interestingly, the technique selected for this evaluation is relatively simple in that it does not rely on overly complicated mathematical or signal processing techniques, such as spectral analysis, autocorrelation or cepstrum analysis, all of which were used in preliminary investigations for this study. Yet its results surpass the preliminary findings suggested by those advanced methods. As a conclusion, this technique is an illustrating testimony to the value of exploring the human evaluation process and using it directly to supplement or replace more orthodox signal processing techniques.

Quality Evaluation for the Gas Metal Arc Welding Process

A good GMA weld is not as strongly characterized by a dense, regular ripple pattern, as the GTA weld is. In many cases a perfectly good GMA weld does not show a distinct trace of surface ripples. Thus, an alternative strategy was developed to evaluate welds from the GMAW process.

Figures 23 and 24 show good quality and poor quality GMA welds, respectively. Generally, good quality GMA welds are uniform and contain little or no artifacts on the bead surface. Furthermore, the bead width is relatively uniform along the length of the bead. In contrast to this, a poor GMA weld typically shows signs of spatter, irregular fusion line with the parent metal on the sides of the weld, and various discontinuities on the weld surface. The human quality assessment was based on the tendency of a good GMA weld to have a very uniform geometry and surface appearance along the longitudinal axis of the weld. A weld which is excellent in these characteristics was given the rating of nine. This weld is illustrated in figure 23. A weld which was performed in the presence of severe surface contamination, which resulted in significant variations in the height, width, and contour of the weld bead was given a rating of two. This weld is shown in figure 24. A GMA weld performed without shield gas, which resulted in severe geometry variation, along with extensive porosity, was given a quality rating of one. Between these extreme cases of quality, a series of GMAW welds were evaluated and assessed a quality rating according to the degree of bead uniformity (geometry and appearance) along the weld axis.

Figure 23. A photograph of a good quality GMA weld bead.

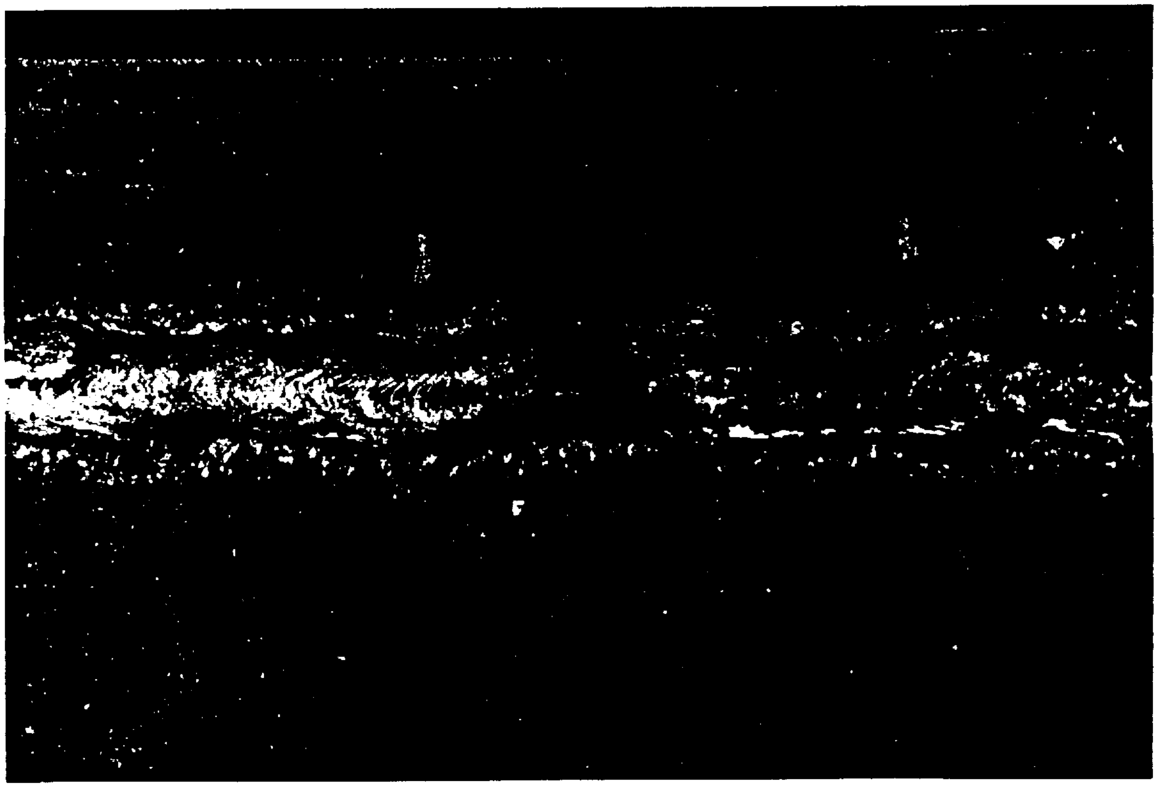


Figure 24. A photograph of a poor quality GMA weld bead.

The general approach used to evaluate GMA welds was to examine the variance of weld profiles, sampled across the weld bead in various locations. A high quality weld would generally yield small variance in the weld profiles, while low quality weld profiles would vary substantially, as irregularities and various discontinuities are encountered in the distinct profile scans. Unlike the GTAW analysis, the GMAW evaluation is primarily based on the horizontal integration technique, which was discussed earlier, in the section on GTAW evaluation. The computerized GMA weld quality evaluation technique is summarized in figure 25. The video image of the bead surface is sampled, digitized, and stored in computer memory. During system development, two aspects of the weld bead were analyzed. The main aspect was the variance of the weld profile, as examined in different locations along the weld. A segmented horizontal integration technique was implemented for this. The second aspect of the weld inspection was analysis of brightness variations along the bead edges. Generally, it may be anticipated that a good weld has more uniform edges than a bad one. For this purpose a full horizontal integration was used to determine the edges of the bead, and then a vertical integration was used to examine the regularity of the bead edges. This second aspect of the weld inspection did not always turn out to be as reliable as the first one, i.e., the segmented horizontal integration followed by the variance analysis. Nevertheless, it is mentioned here to illustrate the various approaches tested for evaluation purposes.

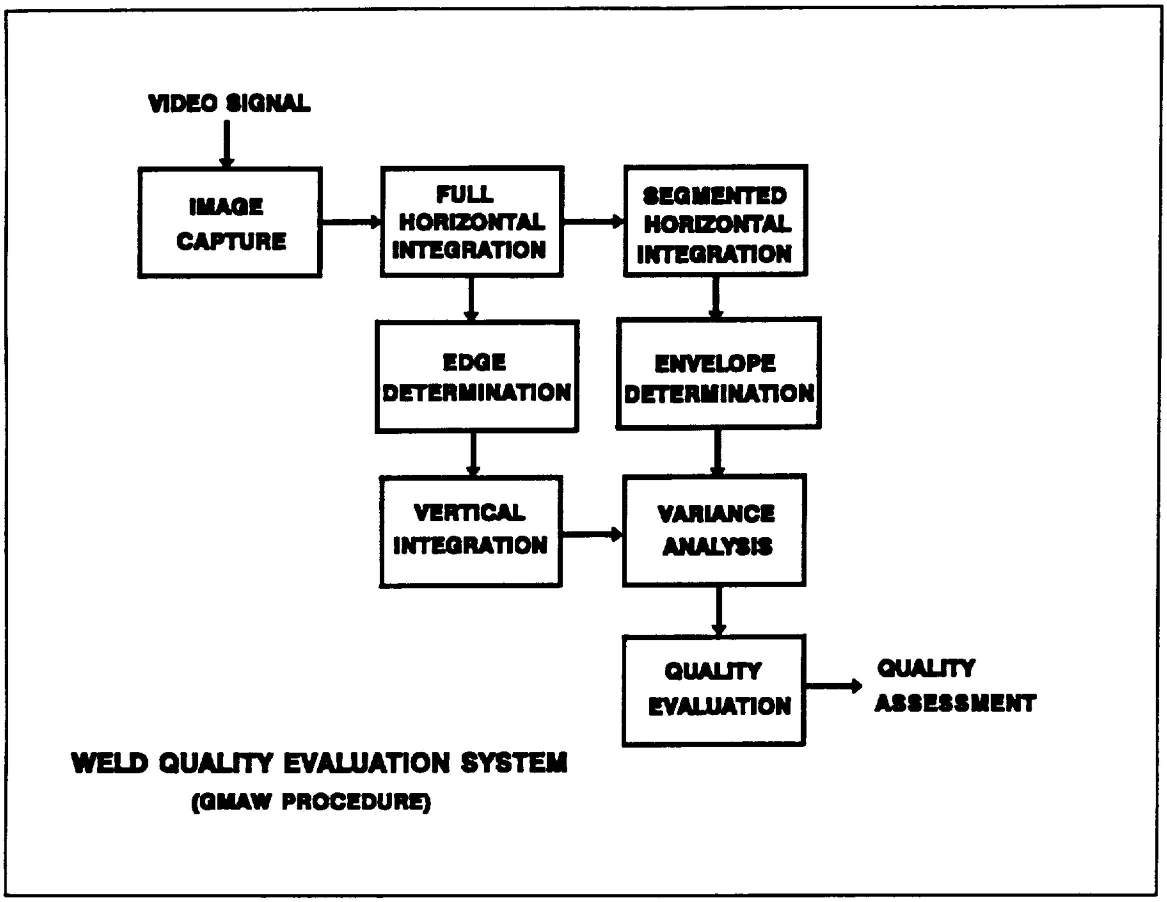


Figure 25. An overview of the process developed for evaluating GMA weld quality.

As for the GTAW evaluation, the general approach for the GMAW process is illustrated through examples. In this case five welds were examined, each one assigned a quality number in the range from 0 to 10 by a human inspector. The selected welds were evaluated with quality indicator values of 2, 4, 6, 8 and 9, and these numbers are embedded in the file names containing image data from these welds. For example, image file GMAW6H1 contains data from a GMA weld evaluated by a human to be of quality 6. The file contains a histogram that was scanned horizontally by the image analysis system (thus the letter H), and the file represents scan number 1 of this weld. The data from this file is illustrated in figure 26. As explained in the chapter on the software system, horizontal integration essentially averages to a specified degree the pixel intensities in the image. The figure therefore illustrates a filtered image of the weld profile, as viewed along the weld axis.

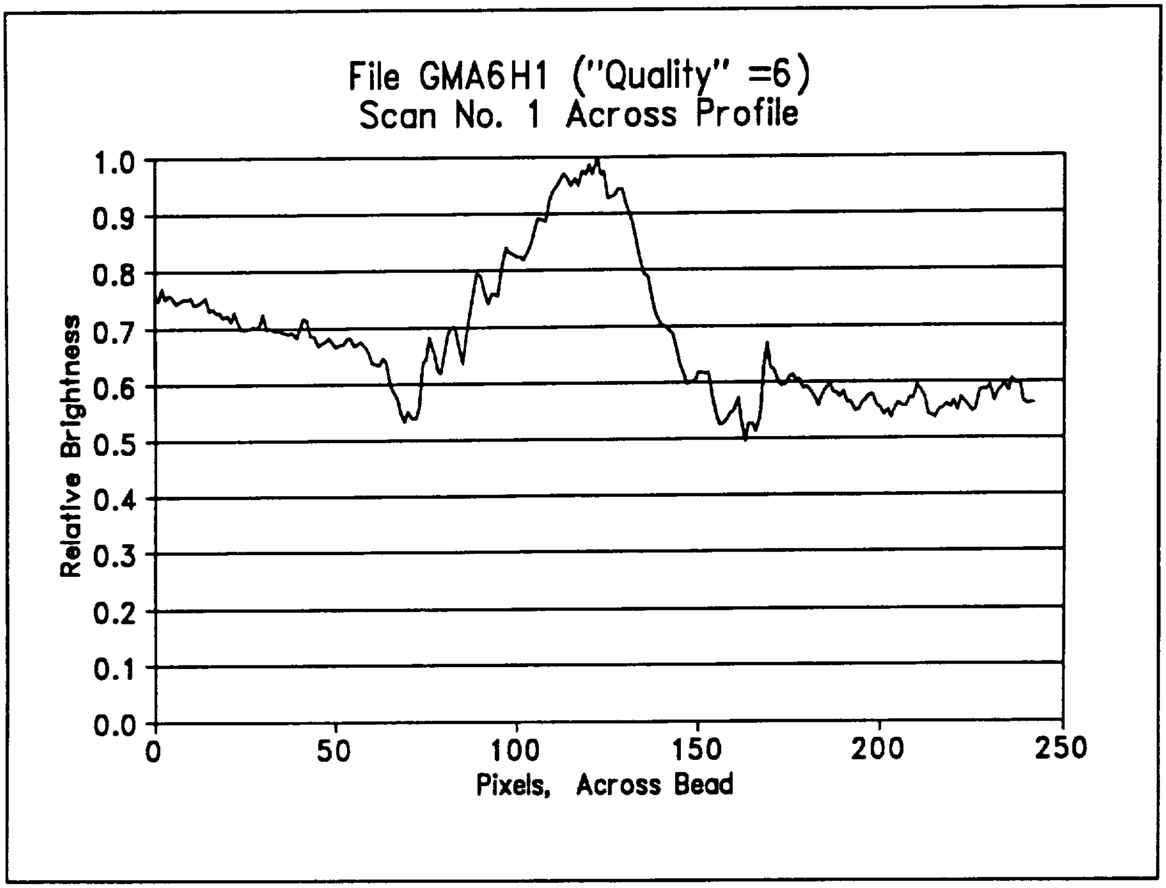


Figure 26. Horizontal integration scan No. 1 of weld GMA6.

While figure 26 shows one scan across the bead, the profile variance along the bead length was assessed by taking a total of 5 scans, each of 16 pixels width. Figures 27 through 31 show the result of this where, in each case, the same weld has been scanned at five different locations and the resulting scans are all graphed together in the same illustration. The similarity between the individual bead profiles obtained from the same bead, at different locations, may be used as an indicator of the uniformity of the weld and thus its overall quality.

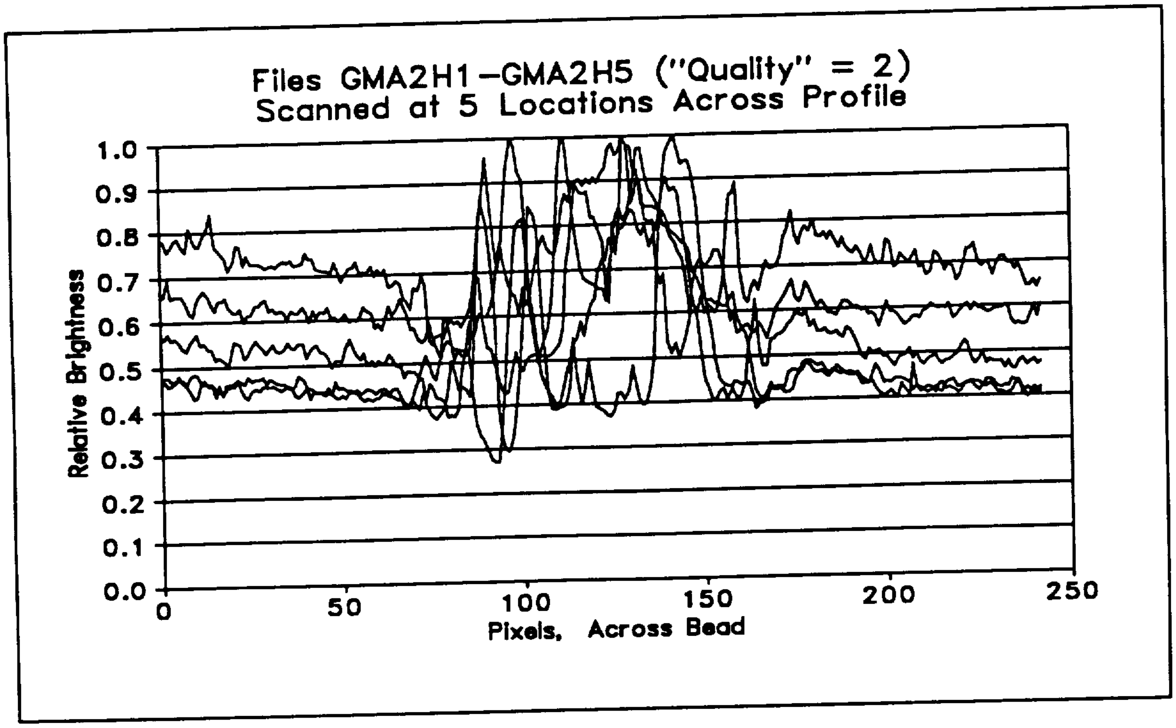


Figure 27. Horizontal integration scans of weld GMA2.

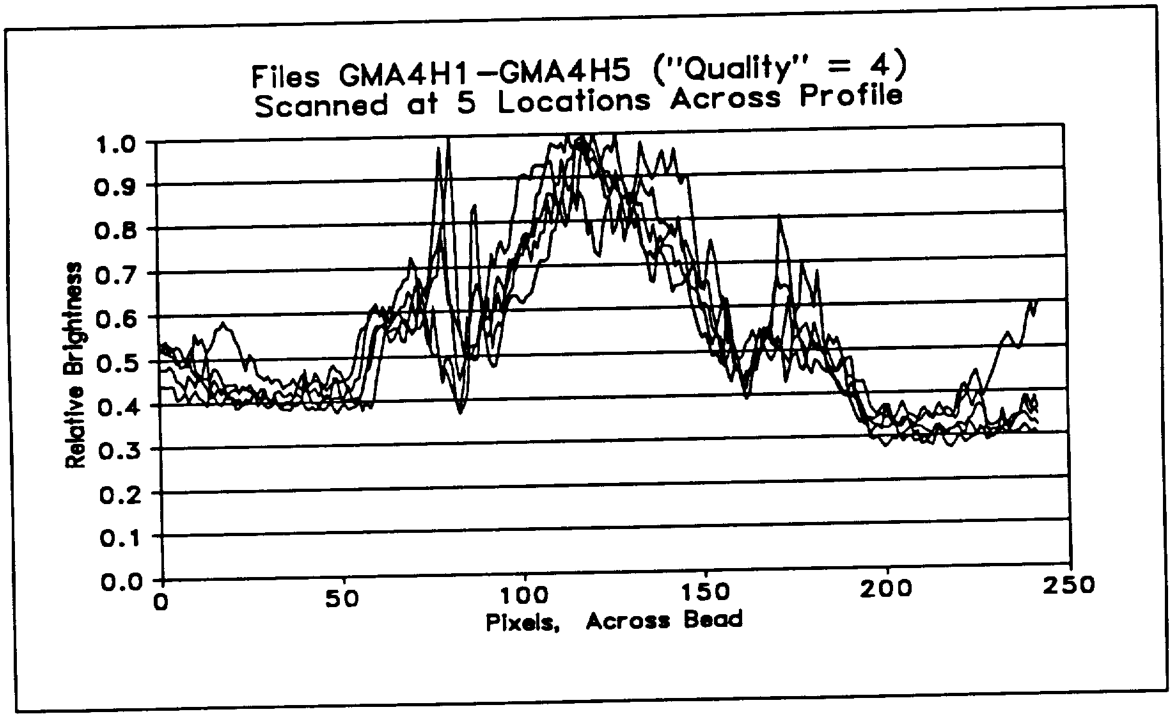


Figure 28. Horizontal integration scans of weld GMA4.

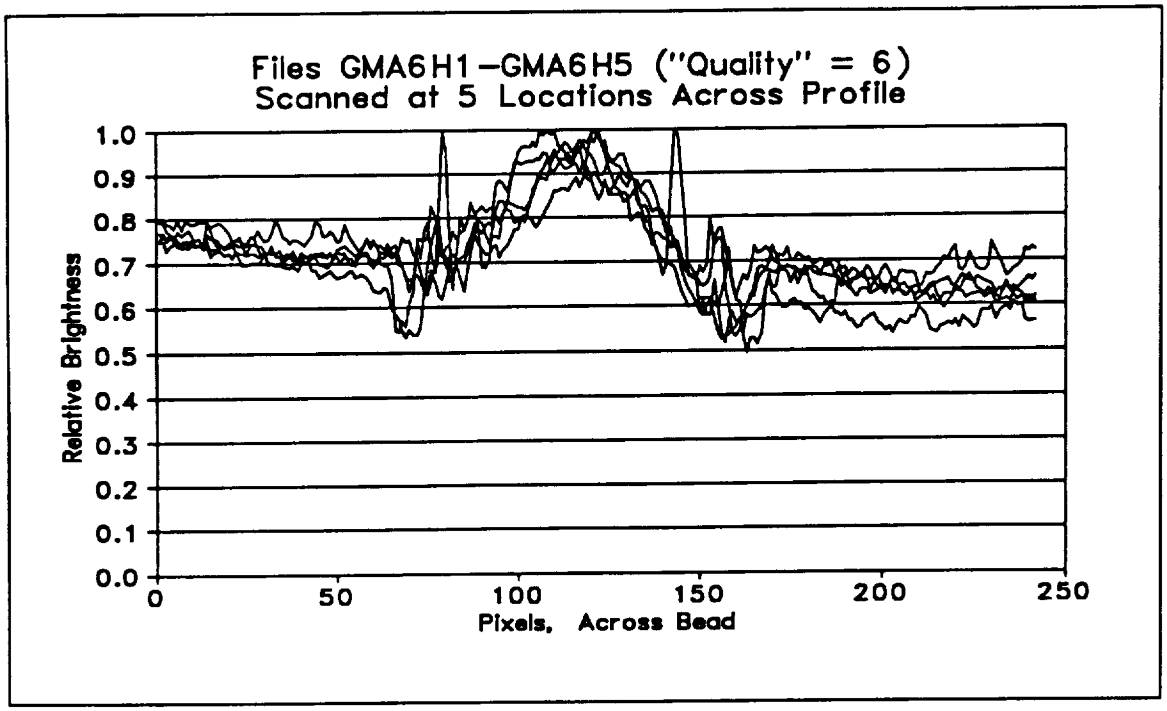


Figure 29. Horizontal integration scans of weld GMA6.

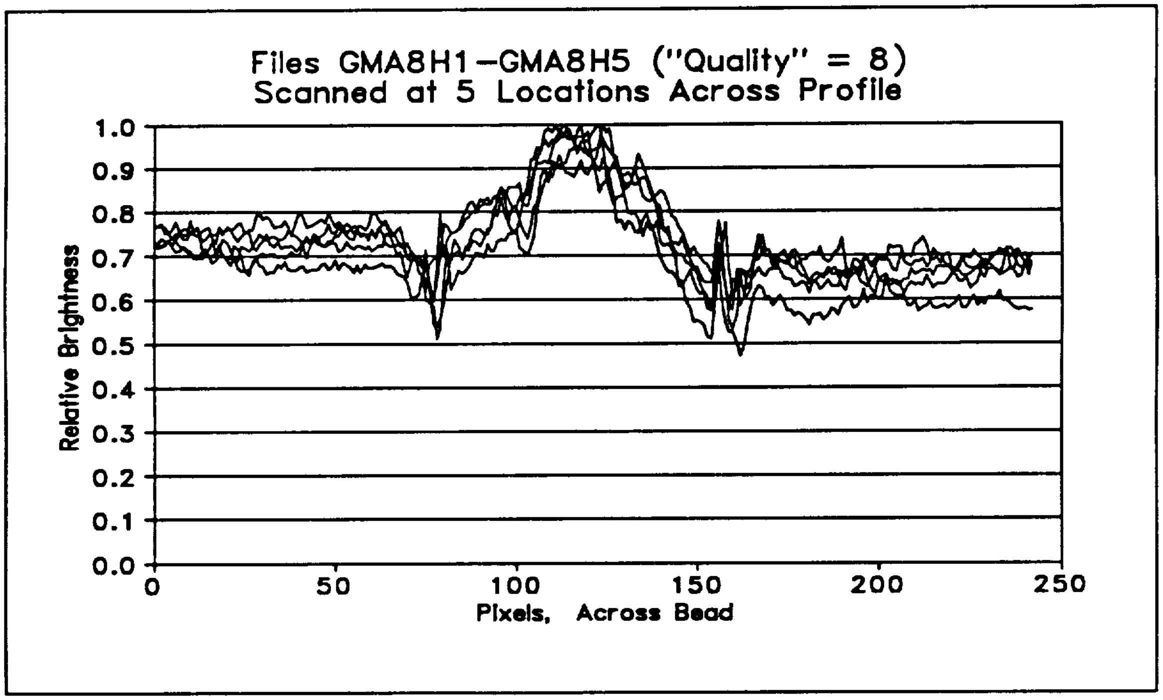


Figure 30. Horizontal integration scans of weld GMA8.

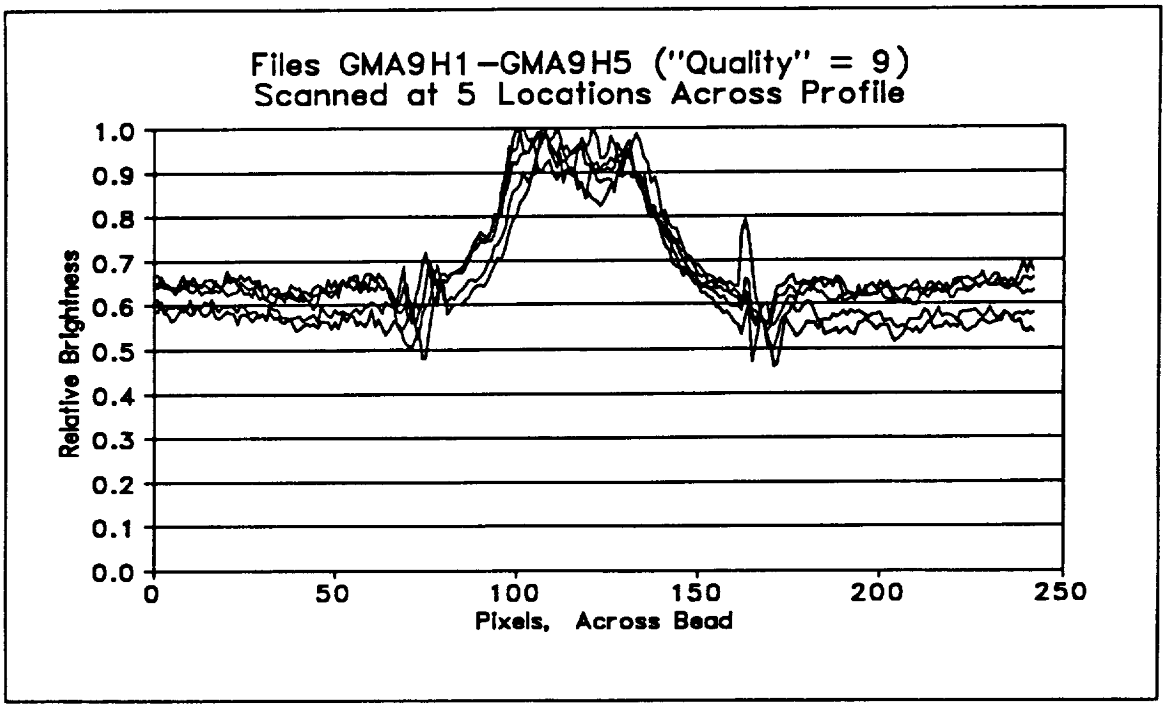


Figure 31. Horizontal integration scans of weld GMA9.

At least two approaches may be taken to quantify the similarity (or lack thereof) of the five scans. One is to measure the envelope of the ensemble of scans at each location, i.e., measure the difference between the maximum and the minimum intensities. The other approach is to calculate the root-mean-square (RMS) variance of the five points of each location. Both methodologies were implemented and tested. Generally, it was found that they yielded similar results. Whenever the envelope measure was large, so was the RMS measure, and vice versa. The calculated envelope differences and RMS variances were determined for each of the tested welds and the results are illustrated in figures 32 through 41.

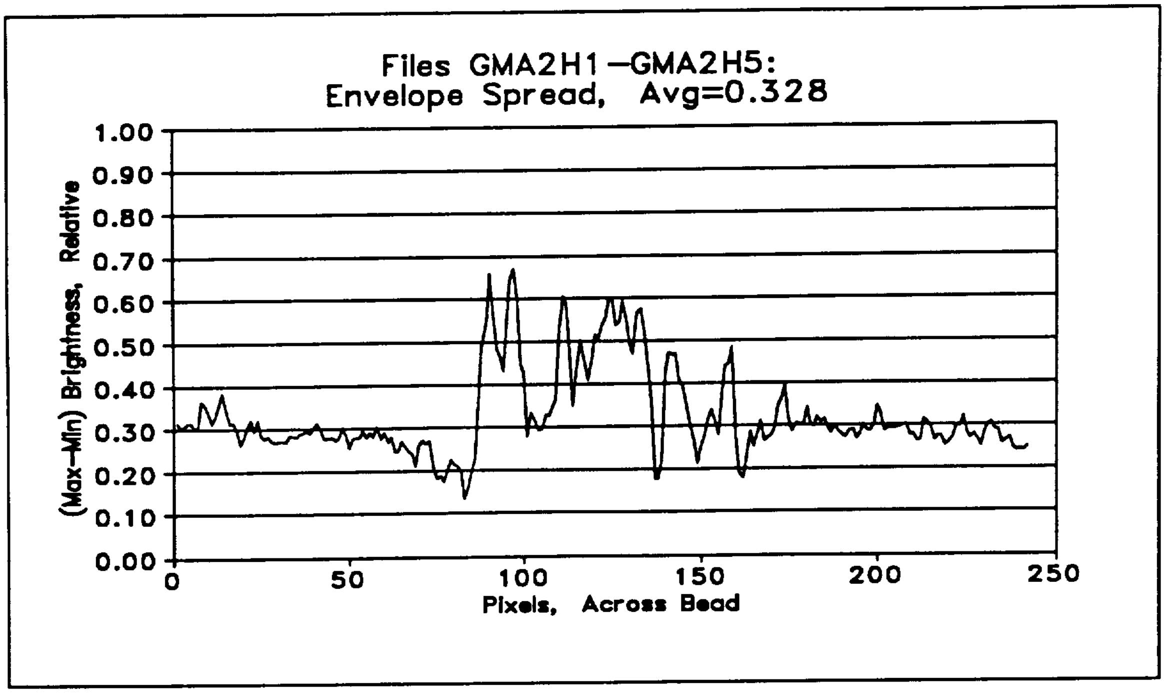


Figure 32. The envelope difference of scans across weld GMA2 (human quality evaluation = 2).

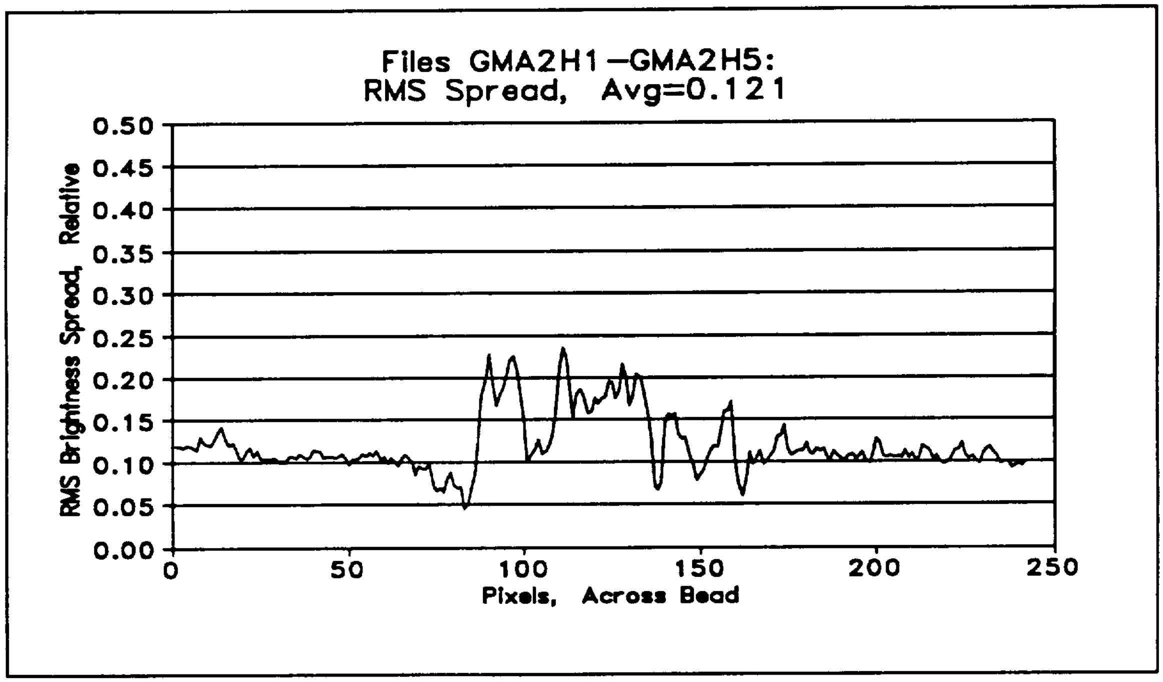


Figure 33. Root-mean-square variance of scans across weld GMA2 (human quality evaluation = 2).

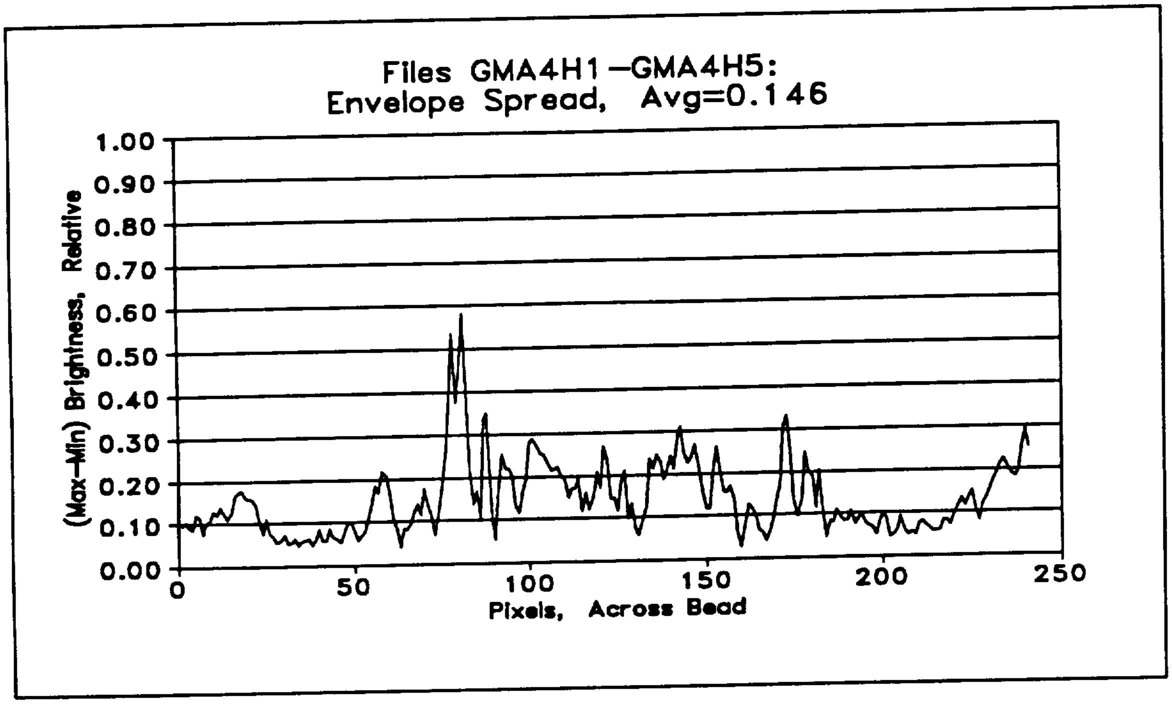


Figure 34. The envelope difference of scans across weld GMA4 (human quality evaluation = 4).

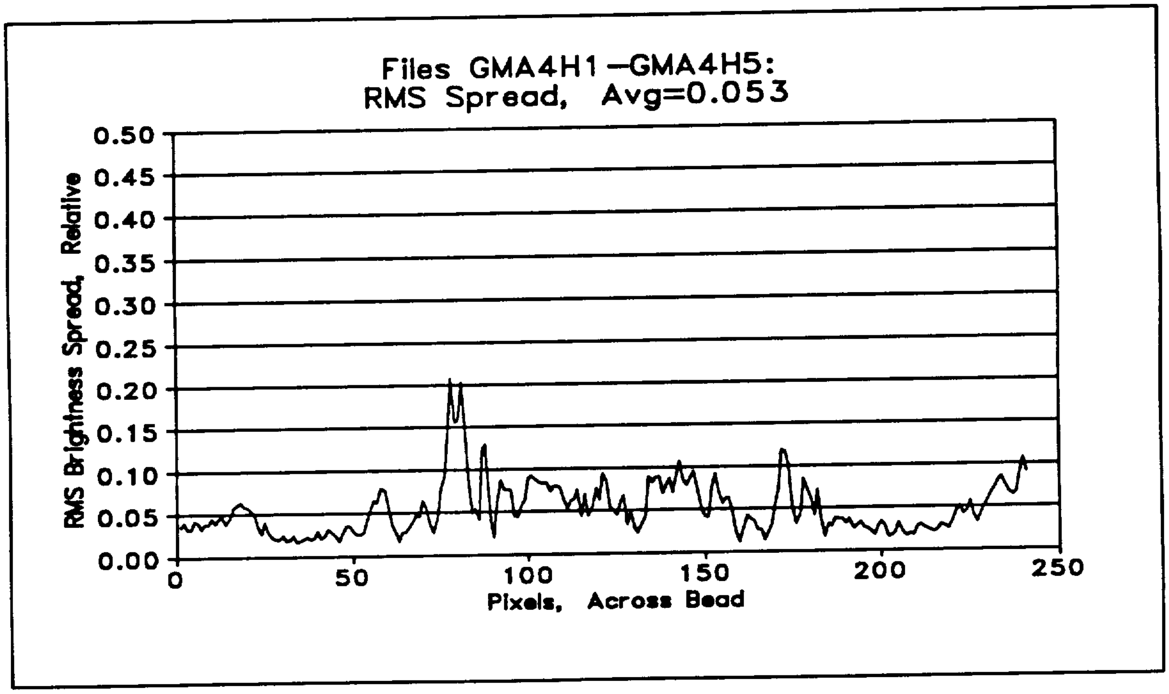


Figure 35. Root-mean-square variance of scans across weld GMA4 (human quality evaluation = 4).

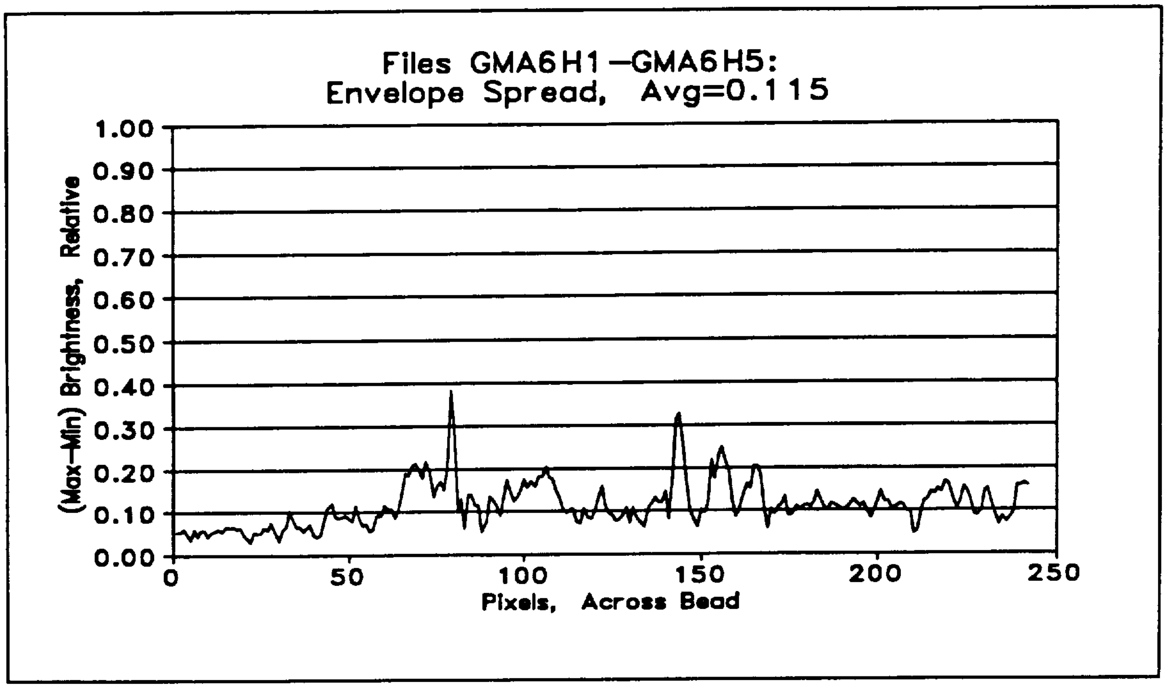


Figure 36. The envelope difference of scans across weld GMA6 (human quality evaluation = 6).

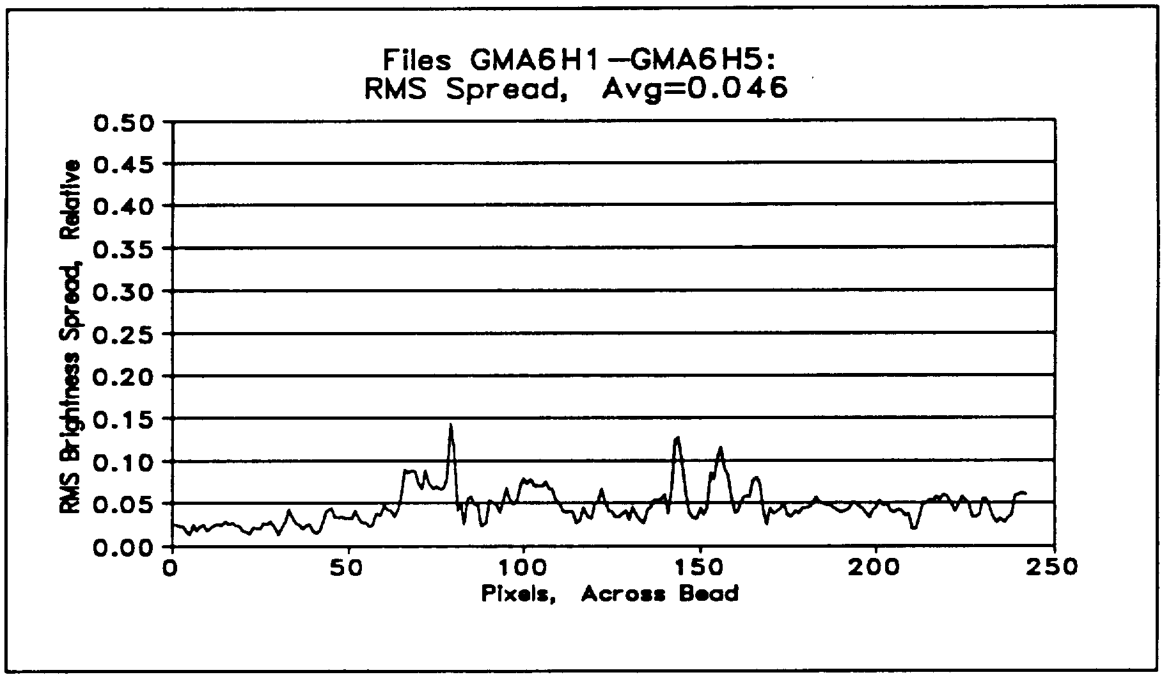


Figure 37. Root-mean-square variance of scans across weld GMA6 (human quality evaluation = 6).

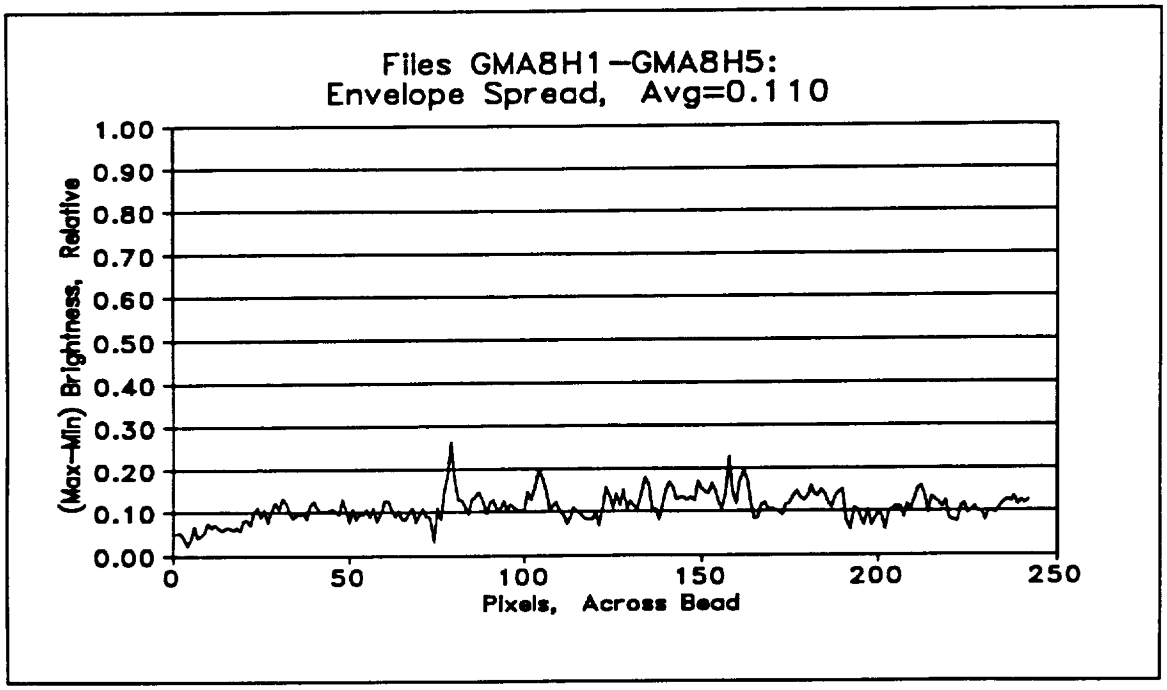


Figure 38. The envelope difference of scans across weld GMA8 (human quality evaluation = 8).

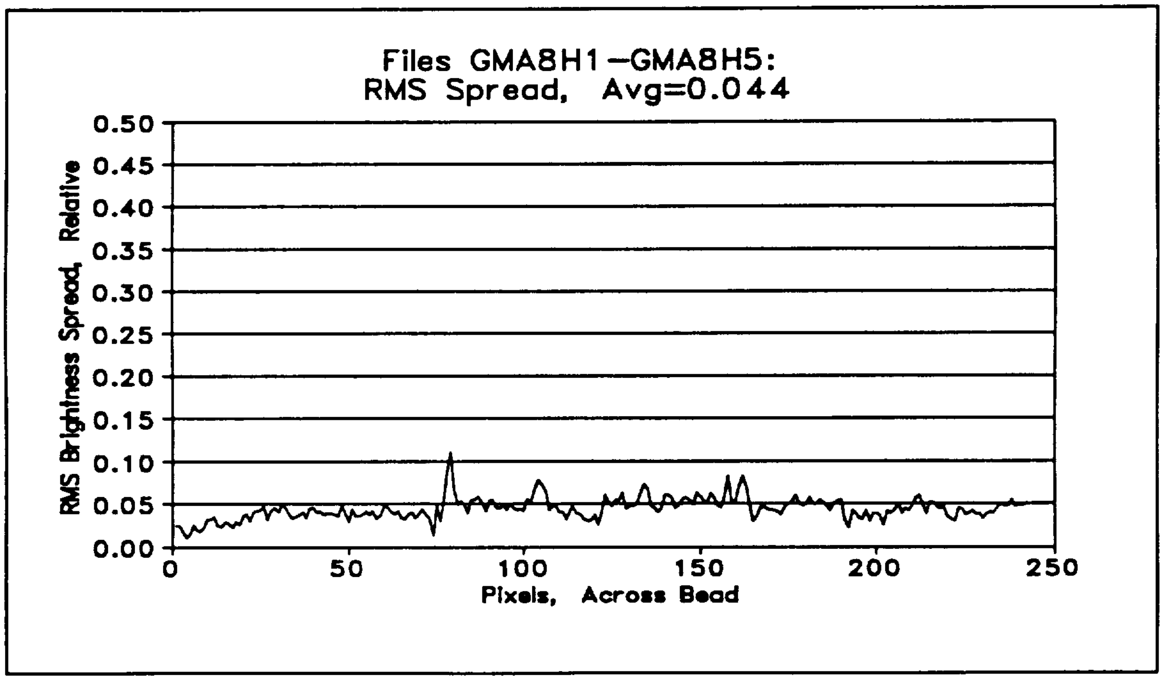


Figure 39. Root-mean-square variance of scans across weld GMA8 (human quality evaluation = 8).

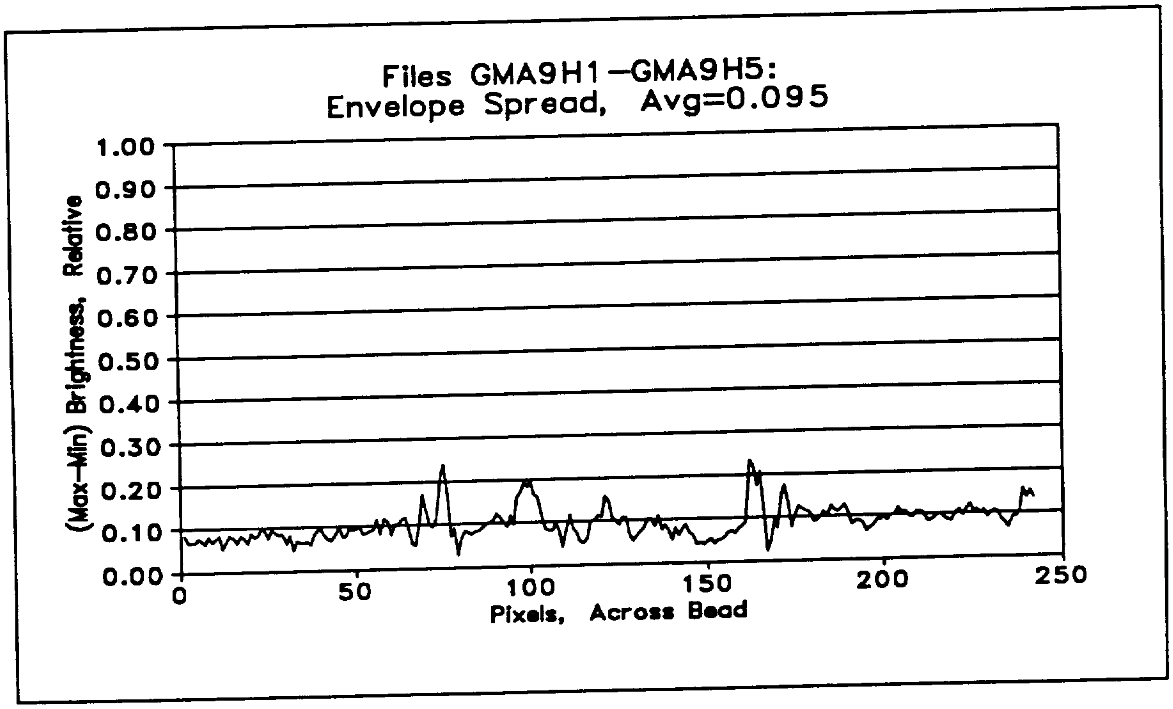


Figure 40. The envelope difference of scans across weld GMA9 (human quality evaluation = 9).

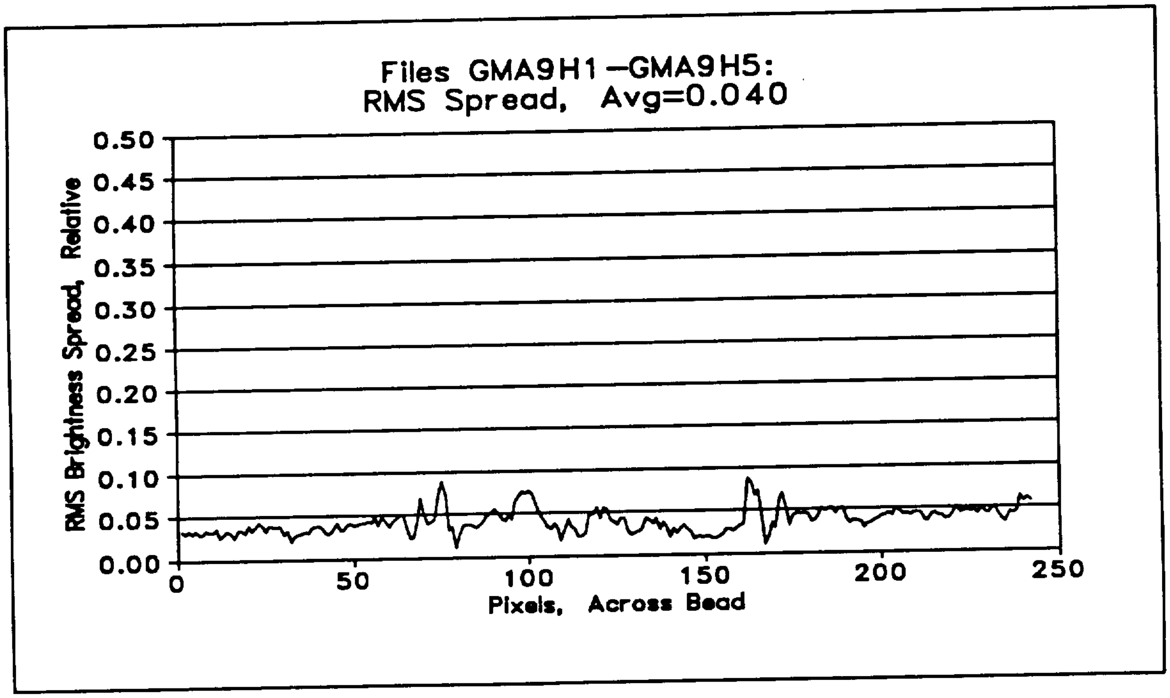


Figure 41. Root-mean-square variance of scans across weld GMA9 (human quality evaluation = 9).

The average of these differences (envelope or RMS measures) yields a indicator of the overall dissimilarity of scans within each weld. The average of each measure is printed at the top of each of the preceding graph. It may be seen that the envelope difference and the RMS variance generally decrease for improved weld quality, as expected. Rather than using only one of the variance measures, both may be combined into a quality indicator of the form:

$$Q = 1 / [(\text{avg envelope spread})+(\text{RMS variance})] \tag{10}$$

This quality indicator is illustrated in figure 42 as a function of the human-assigned quality values. As for the GTAW case, match of the actual values is not important, but rather consistency so that comparison between any two welds yields the same result as a human would come to.

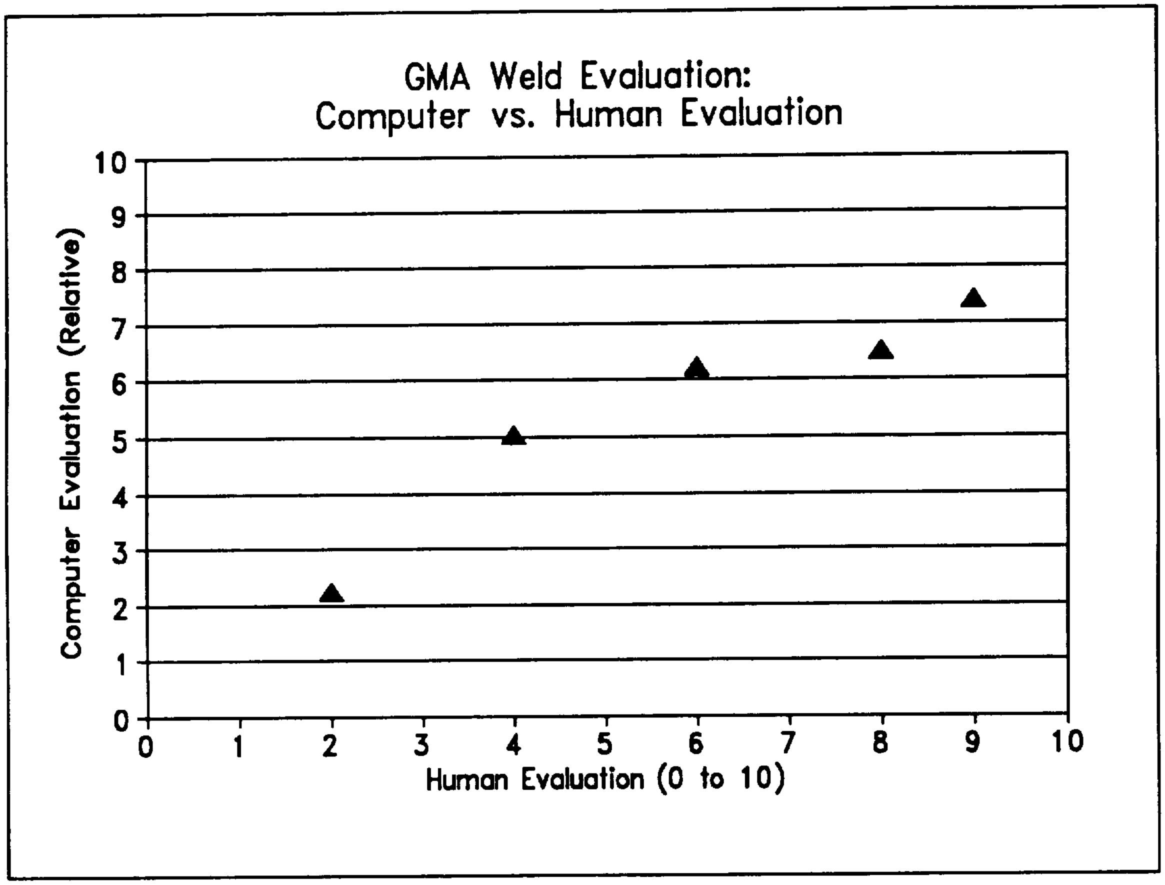


Figure 42. The GMAW computer quality indicator as a function of the human-assigned quality values.

Quality Evaluation for the Variable Polarity Plasma Arc Welding Process

The third welding process of interest for this work is the VPPAW process. This is a relatively new welding process, which is increasingly used by NASA as well as in commercial applications.

The VPPA weld appearance is to a large extent similar to that of the GMA weld. Unlike the GTA weld, which is characterized by a ripple pattern along the bead, the VPPA bead surface usually has negligible or no surface ripples at all. The weld edges, however, may become irregular under certain conditions, a characteristic that is shared by the GMAW process. Actual VPPA weld samples were not available for this work to test the GMAW quality assessment method on VPPA welds, but the method may be assumed to be equally applicable to both processes. An alternative approach is presented here, however, which is based on earlier research [10] in which the VPPA weld bead profile was analyzed with a laser scanner for symmetry and other features that may be used to assess weld quality.

Figure 43 illustrates the physical components of the welding process. The welding torch moves along the joint at a constant speed. In this case a laser scanner is used to obtain the weld profile information. Unlike a camera, which essentially records the light patterns resulting from reflections and shadows from the bead surface, the laser scanner determines the distance to the bead surface. Thus, each scan of the laser yields the true height profile of the bead along the scan across the weld. Through repeated scanning across the bead, as the scanner moves along the weld with the torch, this method may be used to record the true three-dimensional topography of the bead. For the quality control technique illustrated here, however, individual scans are obtained and processed separately. Each profile scan is routed to a neural network that analyzes the profile and determines specified characteristics, such as its symmetry, dimensions, and other desired features. These characteristics are used finally to assess the quality of the weld.

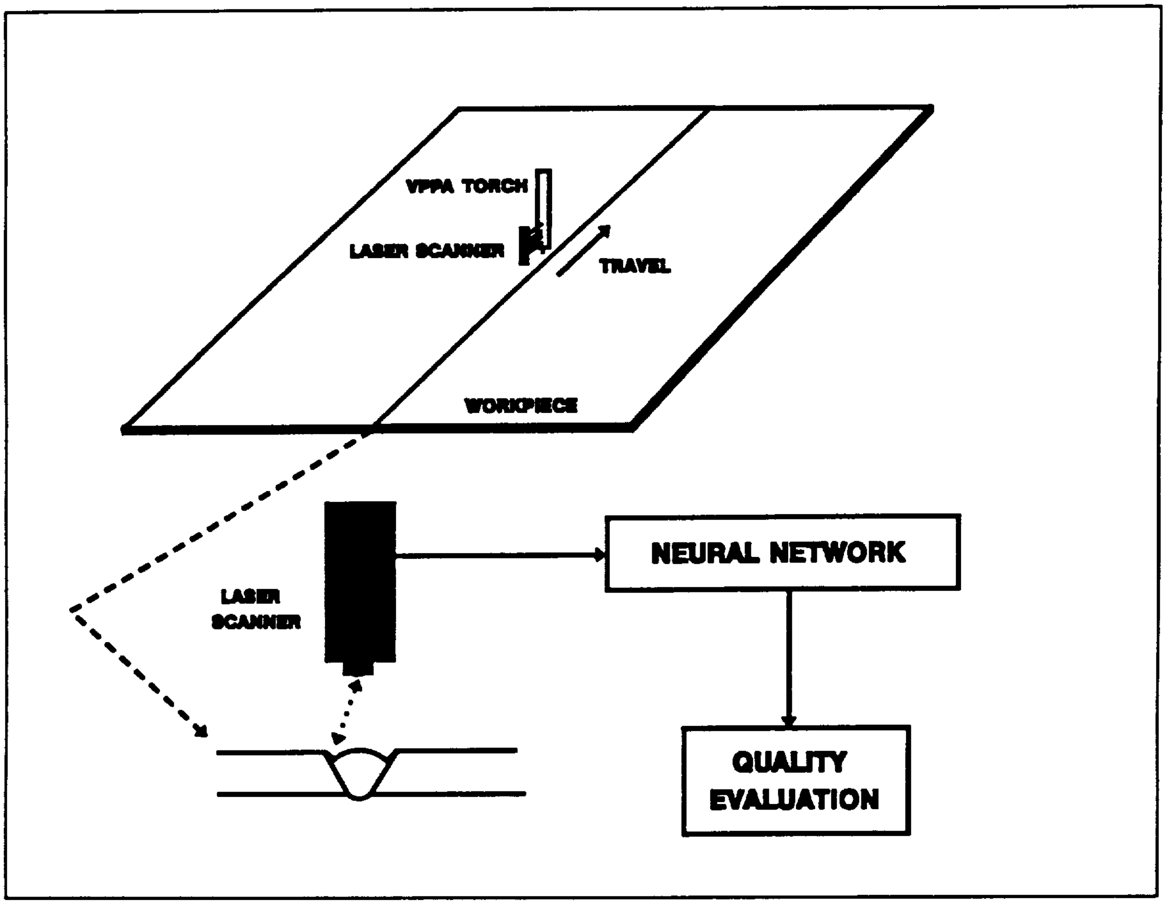


Figure 43. The VPPAW process and the components used to scan and analyze the weld profile.

The weld quality obtained with the VPPAW process may be assessed with a similar method as the one used for GMAW, where scans across the weld bead were compared for irregularities. Figure 44 shows a photograph of a typical high quality VPPA weld cross section. Sequential laser scans of the top of this weld can be used to reconstruct the three-dimensional weld surface profile, resulting in similar set of data as used for the GMAW process.

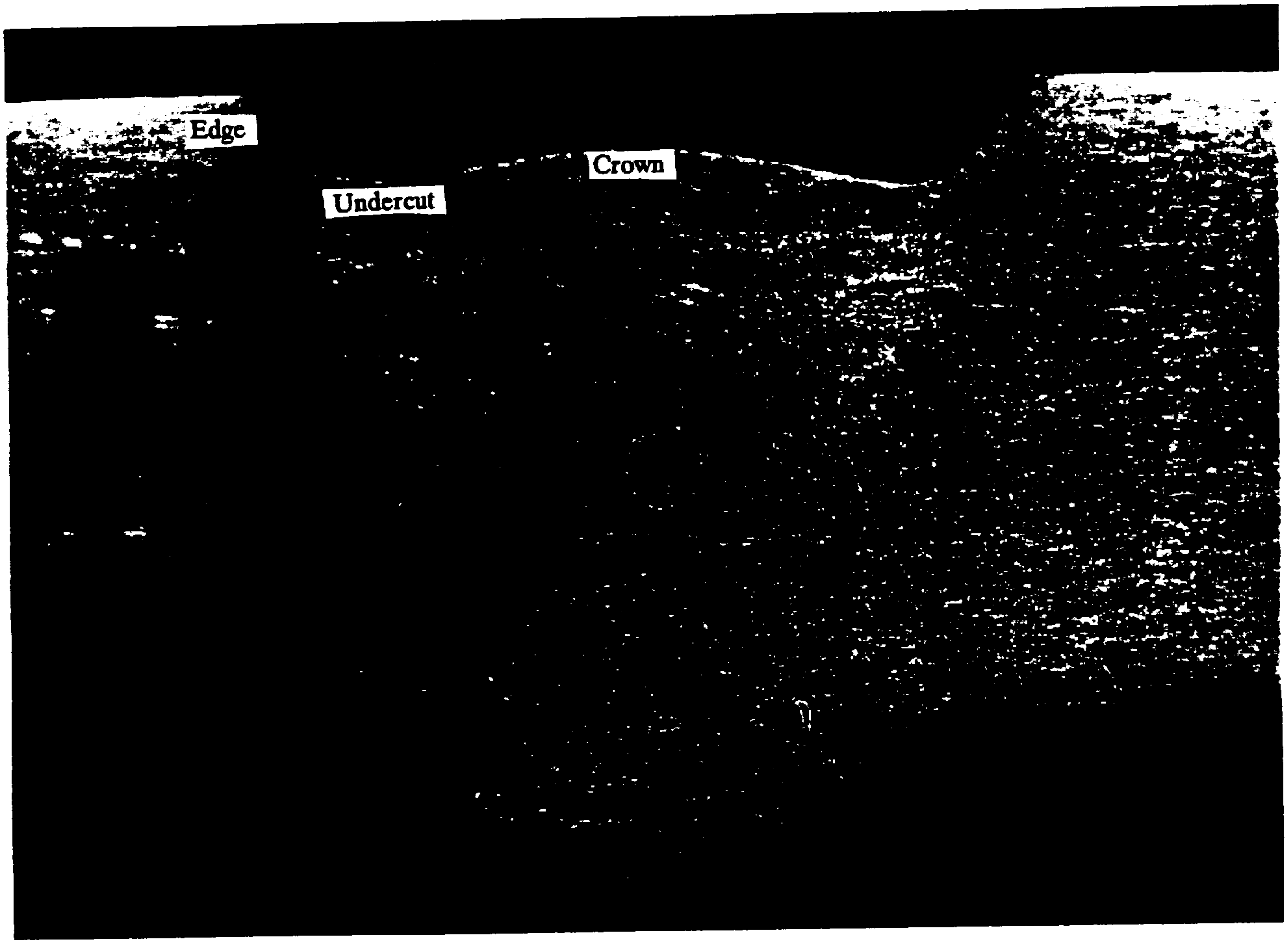


Figure 44. A photograph of the cross section of a typical VPPA weld.

The general VPPAW evaluation process is outlined in figure 45, assuming that the laser scanning method is used. The profile data from a single laser scan is sampled and recorded. A neural network is trained off-line with a part of the available data, so that it may analyze general features of the bead profiles. The results yielded by the neural network are used to evaluate the VPPA weld for various quality measures, such as bead symmetry, dimensional regularity, etc. Any number of quality assessment criteria may be constructed to utilize the results from the neural network. For example, the offset of the crown peak from the true center between the undercuts or the bead edges may be used as an indicator of degraded quality. The same may be applied to mismatch in the depths of the undercuts. These features will be discussed further in the following paragraphs.

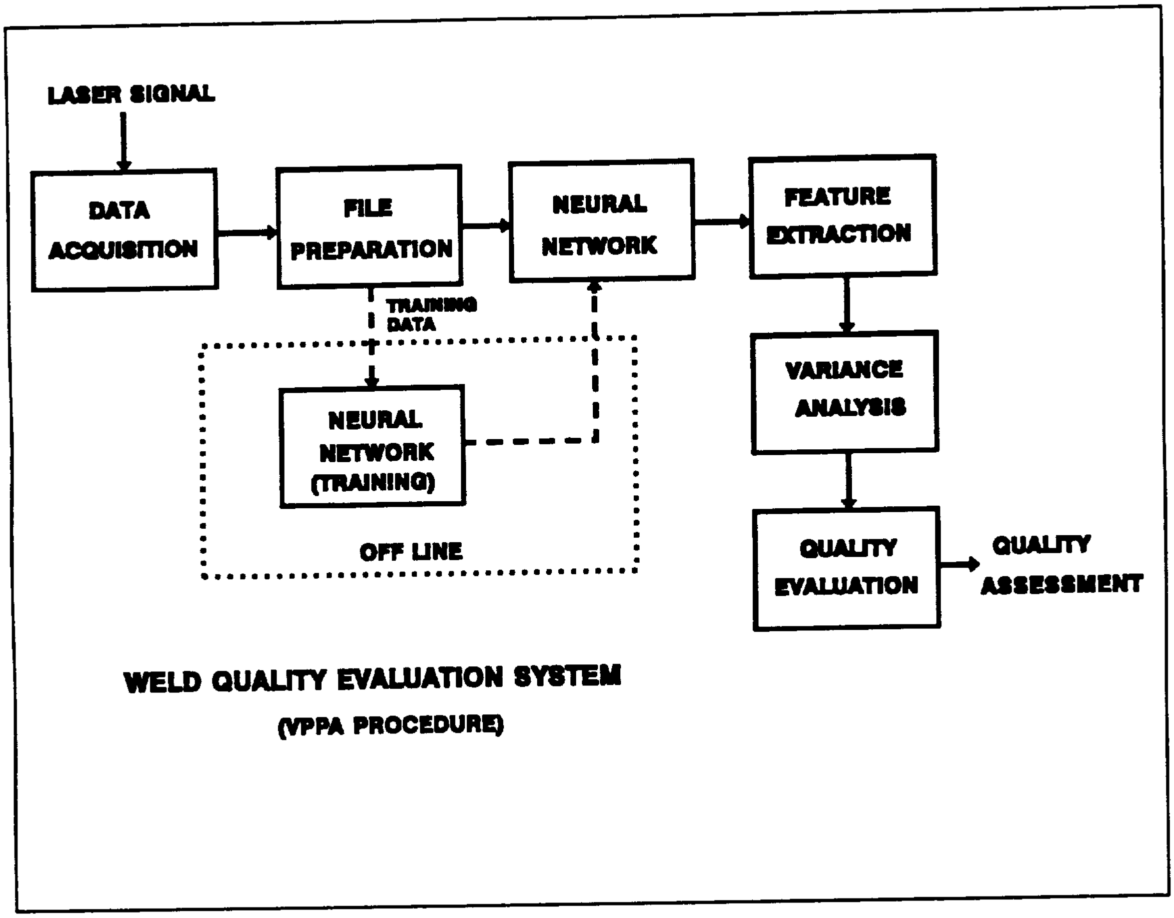


Figure 45. The procedure for VPPA weld evaluation.

Using this approach, several scans may be made across the VPPAW profile and the resulting profile data may be analyzed for variance, which in turn may be used as an indicator of weld irregularity. In addition to this technique, which was discussed in the section on GMAW weld evaluation artificial neural networks may be used to determine the locations of key features of the weld profile, such as its crown, the undercuts and the bead edges. The neural network system used to analyze the digitized weld profiles is outlined in figure 46. The height values of the digitized profiles were entered into the 60 inputs of the network, which was trained to locate the five features of the profile and present these locations at the outputs.

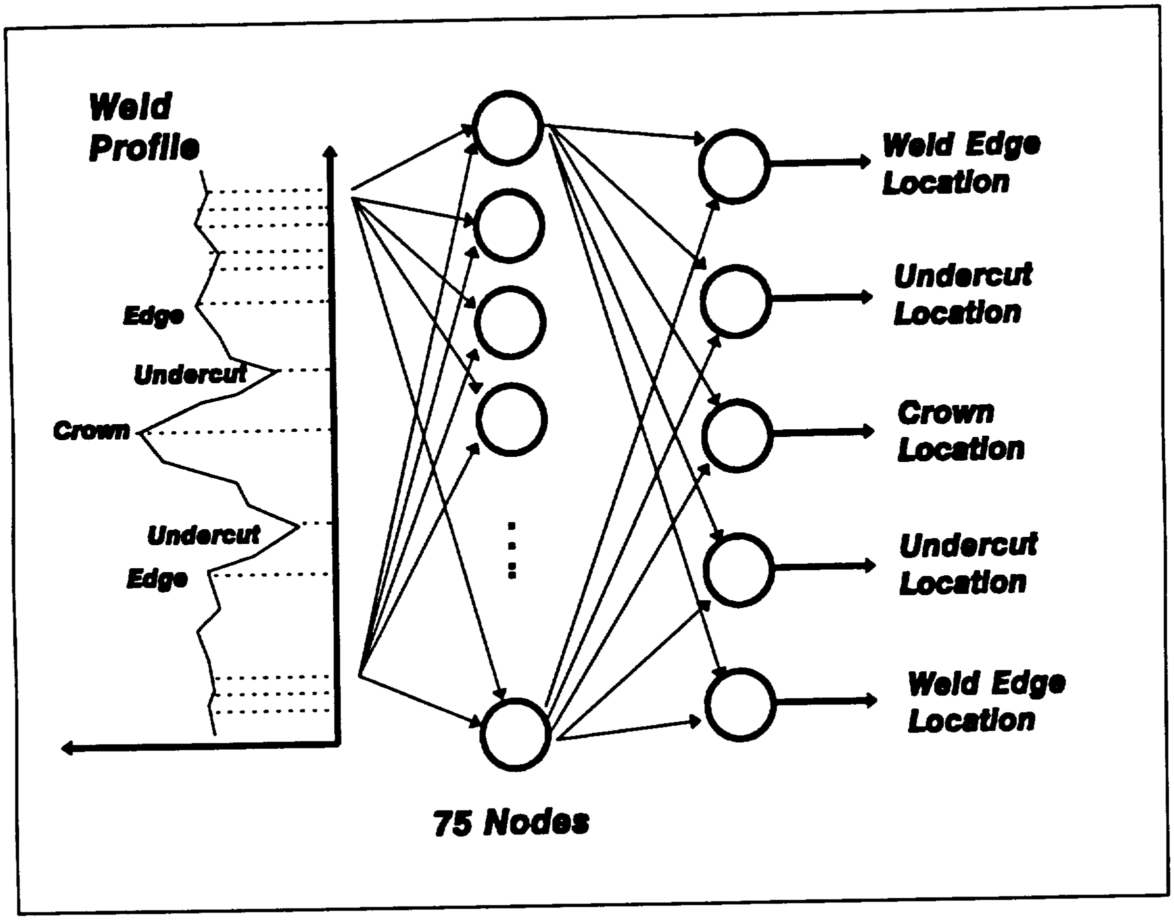


Figure 46. A neural network that locates the crown, undercuts and weld boundaries from weld profile data.

Figure 47 illustrates a typical cross sectional weld profile. The objective of the neural network was to find the locations of the weld crown, the left and right undercut, and the left and right weld boundaries. To the human observer, these features are relatively clear, with the possible exception of the weld boundaries. The locations of these features are indicated on the figure with the arrows marked "N". For comparison, an alternative system, presently used at NASA, located the same features as shown with the arrows pointing from below the profile trace. In this case the present system failed in correctly identifying the right weld boundary, while the neural network succeeded. In most cases the present system performed satisfactorily, while occasionally it gave incorrect results. The errors usually occurred in determining the weld boundaries, and these errors could be significant. For comparison, the neural network performed somewhat better, on the average, and the worst case errors were always much smaller than those of the present system.

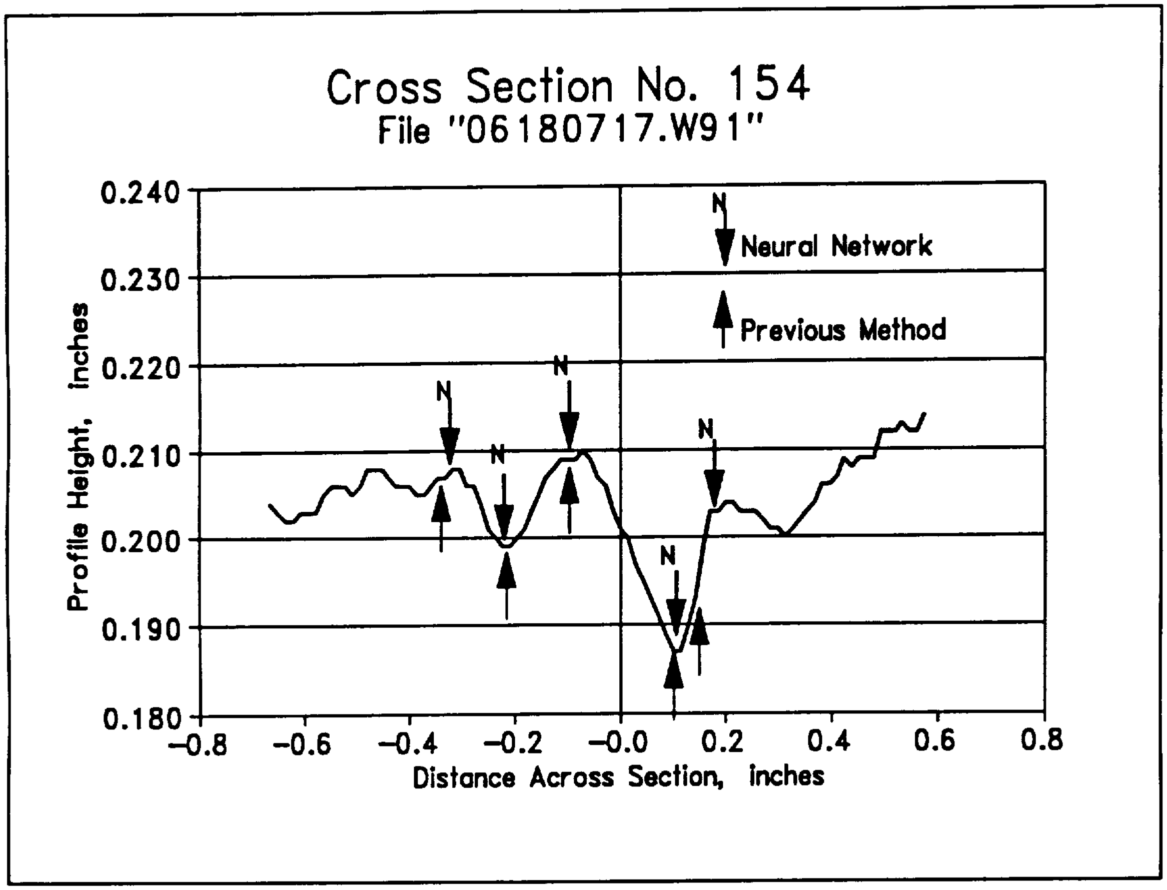


Figure 47. A scanned weld profile, with the weld crown, undercuts, and parent metal boundaries indicated by the neural network and by the traditional system.

A set of 60 bead profiles were used to test and train the neural network. Of the total of 60 profiles, 30 were used for training and 30 were used for testing the trained net. 6 of the testing profiles were sampled for error analysis, and the results are listed in table 1. As shown there, the errors in determining the locations of the bead crown, undercuts and parent metal boundaries are typically less than 1%.

TABLE 1

Sample results obtained with the weld profile analysis neural network. These six weld profiles were not used in original training of the network.

Profile No.:	L-Par Err:	L-Ucut Err:	Crown Err:	R-Ucut Err:	R-Par Err:	RMS Err:
134.17	-0.5%	0.6%	-0.4%	-0.2%	1.8%	**0.9%**
148.17	0.6%	0.7%	-0.3%	-0.1%	-1.1%	**0.7%**
302.17	0.2%	0.6%	1.0%	0.8%	0.0%	**0.6%**
316.17	0.4%	-0.2%	0.3%	0.1%	0.3%	**0.3%**
446.17	1.2%	-0.1%	-0.3%	0.2%	0.4%	**0.6%**
453.17	-0.7%	-0.1%	-0.2%	-0.7%	-0.6%	**0.5%**

From the above examples it may be seen that the neural network can be used as a tool to assess the quality of the individual scans. Particularly, it may indicate lack of symmetry in the weld profile, abnormally wide or narrow beads, or other undesirable features. Further discussion of neural network techniques may be found in [10].

The Integrated Weld Quality Evaluation Approach

The preceding discussion has outlined methods that were developed and tested for weld evaluation. For prototype development, the video based system was used for the GTAW and GMAW applications. In these, the assessment was based upon the quality of the weld beads as determined from the bead images (pixel luminosity values and location) alone. The laser scanner system was used for the VPPAW applications. The process evaluation was based upon actual weld topography as defined by successive laser scans along the weld axis.

Specific examples were given in which video images were used to analyze welds from the GTAW and GMAW processes and output from a laser scanner was used to analyze welds produced by the VPPAW process. As noted in the earlier discussion, however, the video-based, the laser-scanning sensing technique, or a combination of both may be applied to any of the three processes.

The laser scanner process can, of course, be applied to the GTAW and GMAW weld beads. With appropriate training, a neural network could be applied to the task of feature recognition for these processes as well. By proper treatment of the data obtained from the laser scanner, an image of the weld bead may be synthesized. This is accomplished by storage, analysis, and display of successive laser scans as illustrated in figure 48. This synthetic image may then be analyzed and evaluated by numerical algorithms of the type described earlier.

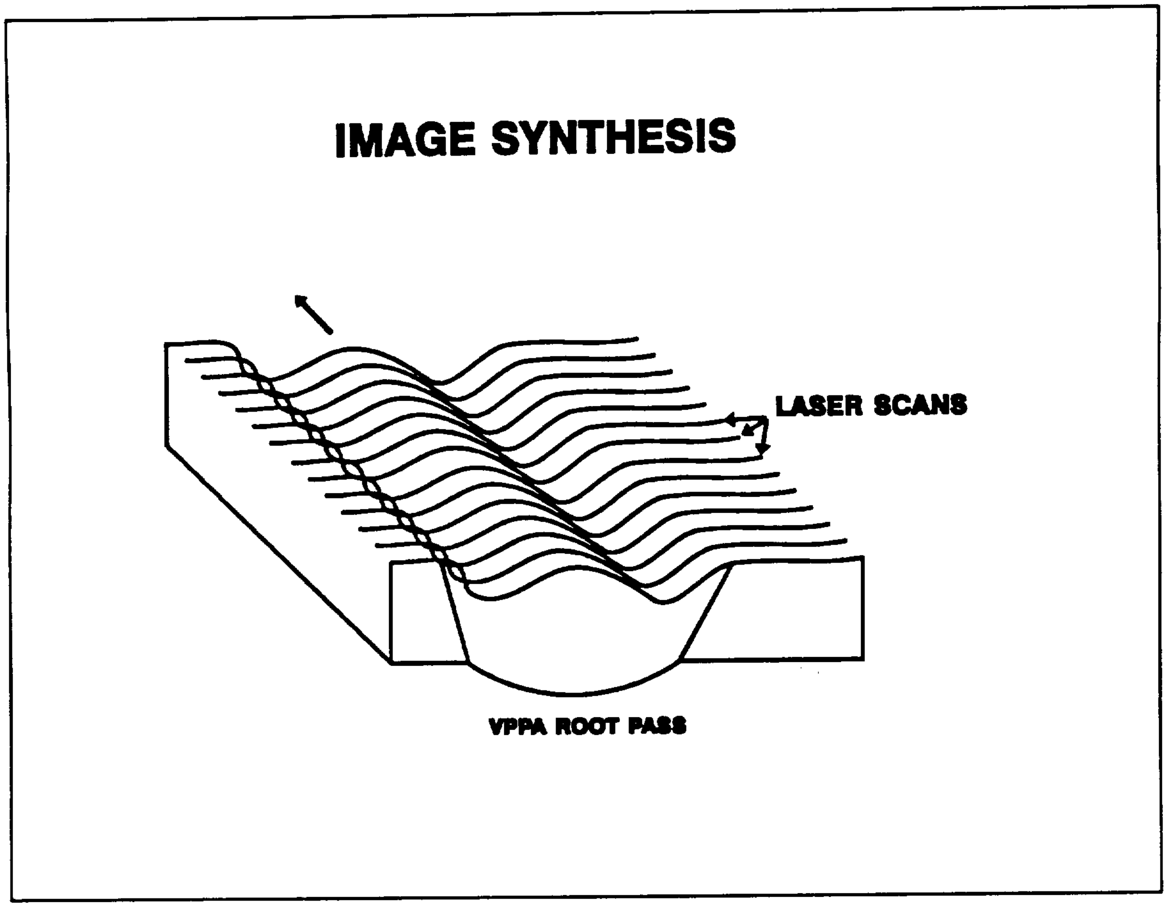


Figure 48. Successive laser scans across a VPPAW root pass may be used to reconstruct a view of the three-dimensional weld bead surface.

An overview of the comprehensive weld evaluation system is shown in figure 49. In this system, the two components of the overall system are shown working in tandem for maximum data acquisition and maximum possible accuracy of quality evaluation. However, it has been demonstrated that either side of the system will provide a high level of quality of the assessment process. For high criticality applications, such as aerospace, both components would be recommended. For less demanding applications, such as in consumer products assembly, the video system alone would often be sufficient, reducing overall system cost and complexity.

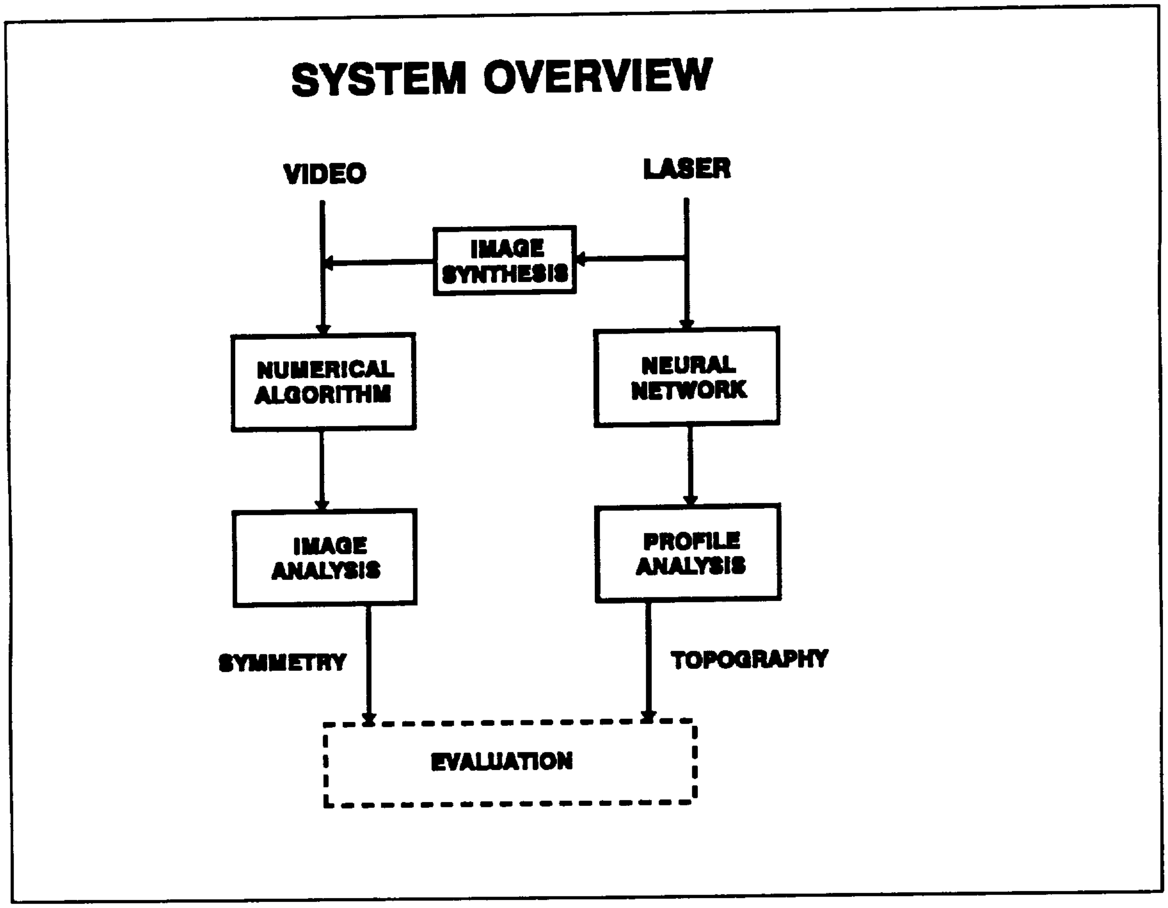


Figure 49. An overview of an integrated weld evaluation system.

This concludes the discussion of weld evaluation approaches applicable for the three selected weld processes: GTAW, GMAW and VPPAW. The main conclusion from this is that techniques can be developed to assess the quality of the weld based on the specified features. Other quality criteria may be built into the weld evaluation system if desired, to supplement further the assessment process.

CHAPTER VI

QUALITY CONTROL SYSTEM PROTOTYPE IMPLEMENTATION

The main components of the prototype system developed for this work are outlined in this chapter. Broadly, the system may be separated into hardware components, which mostly consisted of commonly available equipment. The software, which is the primary deliverable item of this contract, is usable with most IBM-PC compatible hardware, as discussed below. This design choice was made to encourage commercial replication of the system.

Hardware Description

A variety of computer and image processing hardware may be used with the software system developed for this work. The primary requirement is that the system is based on the IBM-PC architecture and that the computer uses the 80286 microprocessor or any of its successors. This architecture was chosen to facilitate use in commercial applications at modest costs. The specific system components which were available at Mid-South Engineering and thus used for this development are outlined below.

The host computer on which the system runs is a Dell System 310 computer, which is based on the Intel 80386 microprocessor and is equipped with the 80387 numeric coprocessor. The computer is configured with 4 MB of extended memory and a 100 MB hard disk. A Dell VGA computer monitor is used with the computer. Its purpose is strictly to control the system through the keyboard or the mouse. Graphics display is provided on a separate graphics monitor. The graphics monitor is made by Electrohome, but any RGB-compatible monitor would work as well for this purpose.

For image capturing and display the Data Translation DT 2871 Color Frame Grabber is used. An optional add-on RGB/NTSC converter, DT 2869, may be used to convert images between the RGB and the NTSC television signal standard if an NTSC monitor or an NTSC camera is used. The frame grabber receives the red, green and blue signals, in addition to the sync signal, from a video camera. Internally, the board converts the RGB signal to a hue-intensity-saturation (HSI) form, where three 512-times-512 byte buffers store the image information. A fourth auxiliary/overlay buffer is located on the board for data manipulation. The DT 2871 board outputs the stored image on the RGB format, where the sync signal is transmitted with the green-channel or may optionally be obtained through a separate connector.

The video camera used for the work is made by Panasonic, model Digital-5000. This camera provides an RGB output signal, in addition to a sync channel. Its resolution is 512-by-512 pixels. The main advantage of this camera over many other models, in addition to the high-quality RGB output, is the standard 35 mm bayonet lens mount. This facilitated the use of economical and widely available lenses for this work, including various close-up configurations which were necessary for welding inspection.

A tablet equipped with light sources and a camera column was used as an experimental setup to record images of weld samples. Both the light fixtures and the camera column were adjustable, permitting the necessary flexibility in adjusting lighting conditions and camera position.

As an alternative or as an addition to inspecting the bead appearance with a video camera, a laser profiler may be used, as discussed in context of the VPPAW process. Using a laser profiler with the camera gives the system an additional degree of reliability, which may be beneficial for exceptionally critical applications. A laser-based weld bead profiler is already available for VPPAW work at NASA.

Software Description

The software developed under this contract and its operation is described in this section. The development of this software constituted a substantial part of the work, after the necessary preliminary research.

The objective of the software was to digitize and process images of welds or other inspected material surfaces for the purpose of enhancing discontinuities for visual inspection. Furthermore, algorithms were developed to assess the overall quality of welds from three distinct welding processes: Gas Metal Arc Welding (GMAW), Gas Tungsten Arc Welding (GTAW) and Variable Polarity Plasma Arc Welding (VPPAW). The quality is expressed on a numeric scale, e.g., from 0 to 10.

To make the software accessible to a large portion of the commercial user market, it was implemented on the IBM-PC platform. Initial work was carried out under the MS-DOS operating system. The limitations of early versions of MS-DOS were, however, soon encountered. Of particular concern was the limited amount of continuous memory that MS-DOS could access for program data. The MS-DOS operating system was limited to a maximum of 640 KBytes of memory. This was a severe limitation, as a typical image of 512-by-512 pixels contains a total of 250 thousand pixels. Multiply this by 3 bytes for each pixel (red, green and blue information or, alternatively, hue, intensity and saturation), and the total memory requirement for the image is found to be about 750 KB. Not only does the raw image exceed the memory space that MS-DOS is designed to handle, but memory for the program itself and various data buffers is needed as well. This limitation may be circumvented by temporarily directing data to disk or expanded/extended memory, but these manipulations call for significant overhead and complications in programming. Thus, the OS/2 operating system was selected for the final implementation. This operating system is not limited by the older MS-DOS system. The downside of choosing a less common operating system, on the other hand, was somewhat higher software price and less availability of software tools. Since system programming started in the OS/2 environment further developments have taken place in the IBM-PC arena. Microsoft Windows, version 3.0, was released, offering the menu-driven interface characterizing the OS/2 with its Presentation Manager. The MS-DOS system evolved to version 5.0, which greatly enhances memory management. As the present work is concluded a new, greatly improved, version of OS/2 has been released and Microsoft Windows NT is about to become available. Either of these operating system would be considered for further implementation and commercial distribution of the image analysis software.

Menu-Driven Software Overview

The OS/2 software delivered at the conclusion of this work is described in this section. It may be operated on IBM-PC compatible computers, using the Intel 80286 microprocessor or any of its successors. At least 2.5 MBytes of memory is required, but 4 MB is highly recommended.

The main window of the software system opens up with a choice of five general menu items: File, Process, Texture, Edge and Help. By clicking the mouse pointer on any of these main selections the user is offered submenus, options or other information. The individual items in the menu system will be described in the following paragraphs. Note that the monitor displaying video images is referred to as the graphics monitor, while the one displaying the menu system and the mouse cursor is the system monitor.

File:

Acquire:

This option allows the user to digitize and store an image from the video camera. The computer enables the video camera connected to the system and its view is displayed in real-time on the graphics monitor. Simultaneously, control buttons are displayed on the system monitor which allow the user to cancel the digitization or continue and digitize the image on the graphics monitor. The image is assumed to be in hue-intensity-saturation form.

Acquire RGB:

This option is similar to the one above (Acquire), except in this case the captured image is in red-green-blue format. This is the default operation for image capturing in the present system.

Open:

Opens an existing image file for view and further analysis, and displays it on the graphics monitor. This is an alternative to the acquire commands which import images from the video camera. The file that is opened is by default designated by the extension ".IMG". The system displays a scroll window from which an existing file (if there is any) can be selected and displayed. The file must have been acquired earlier and saved by the Save command (see next menu item).

Save As:

Saves the image that is currently displayed on the graphics monitor. This may be an original image, as acquired from the video camera, or it may be a processed and altered image. In either case, the system asks for a file name to store the image under. Each image file requires approximately 787 KBytes of storage space.

Set Active Buffers:

This selection prompts the user to select one of the following items:

Intensity: Selects the memory buffer that contains the image intensity information for further processing.

Saturation: Selects the memory buffer that contains the image saturation information for further processing.

Hue: Selects the memory buffer that contains the image hue information for further processing.

Overlay: Selects the auxiliary overlay buffer in the image acquisition board for further processing.

Clear Unused: Clears the image buffers that are not currently selected for processing.

Exit:

Terminates the program.

Process:

Blowup:

This selection prompts the user to select one of the following items:

Enlarge: Enlarges each pixel on the screen by a factor of two in the horizontal and the vertical directions. On first usage, the value of every fourth pixel in the image is copied into three neighboring pixels adjacent to it, resulting in a formation of squares of four pixels with identical values. Successive activations of the enlarging operations work in the same manner, creating squares 4-by-4, 8-by-8, etc., of identical pixel values.

Zoom: Similar as the "enlarge" operation, except interpolation is used rather than copying of pixel values.

Integrate:

This selection prompts the user to select one of the following items:

Horizontal: Integration of pixel intensities along horizontal pixel lines. The result of this operation is a 1-dimensional graph that shows the integral of pixel densities as a function of vertical location in the image. The user can specify a window of any width (selected as the "Number of Lines" option) and horizontal position (selected with a scroll bar), extending from the top of the image down to its bottom. Each line of pixel within this window is integrated, resulting in a measure of the average pixel intensity in that line. This is repeated for all lines from the top to the bottom of the image. By clicking the mouse on the "Refresh" button in this window the user can display a graph of the integrated values. Large values indicate high pixel intensities (bright pixels). The displayed graph is normalized so that the largest integral value is plotted at the top of the graph. The integration graph may be saved as a disk file by clicking the mouse on the "Save" button.

Vertical: This operation is similar to the one for horizontal integration, except integration takes place along vertical lines (sometimes referred to as "columns") in the image, the vertical dimension and location of the integration window may be varied, and it extends between the left and right boundaries of the image.

Histogram:

This selection prompts the user to select one of the following items:

Intensity: Displays a histogram of pixel intensities in the entire image. A low-contrast image would typically have a narrow histogram (not much variation in pixel brightnesses) while a high-contrast image would result in a broader histogram. The user can select how many bins the intensity levels are grouped into. For example, with a total of 256 intensities offered by the image board, a selection of 64 bins would result in pixel intensities 0 through 3 grouped together, 4 through 7, etc. Clicking the mouse on "Refresh" updates the histogram display.

Saturation: This histogram operation is similar to the one for intensity, except a histogram of pixel color saturation levels is displayed.

Hue: This histogram operation is similar to the one for intensity, except a histogram of pixel color hue values is displayed.

Enhance Red:

Selection of this option converts all hues of red in the graphics monitor image into a bright red color. The intensities of all pixels with other colors are set to zero, making the effectively black on the screen. This operation is useful for analysis of images preprocessed with a dye penetrant, which usually is red.

Equalize:

The equalization operation generally increases the contrast in the image displayed on the graphics monitor. The system accumulates a histogram of the original pixel intensity distribution and performs a mapping of it that results in equal distribution of all pixel intensities in the image.

Threshold:

Pixels with intensities below a specified threshold value will be turned off (displayed as black) while those with intensities equal to or above this value are displayed as bright white. The threshold value may be specified between 0 and 255 by the user through a scroll bar on the menu. A click on the "Update" button updates the graphics screen image accordingly.

Average:

This is a smoothing operation which is useful if the image is "noisy" or contains small, irrelevant artifacts. Each pixel intensity on the graphics monitor is replaced with the average of the 3-by-3 square of pixel of which it is the center. The image on the monitor is updated after the averaging. This operation may be iteratively repeated to obtain an increasingly smoothed image.

Inverse:

The "Inverse" operation replaces each pixel intensity with its "opposite" value so that, e.g., all pixels with intensity of 0 become 255, 1 becomes 254, etc. The effect of this is a "negative" of the original picture, similar to what is obtained on negative photographic films.

Texture:

Fourier Transform:

The Fourier transform is a widely used signal processing technique which reveals periodic variation in the analyzed signal. The transform was implemented in the software as a tool to analyze periodic ripple patterns along the weld bead. The result of the Fourier transform of the image is a new image, which is displayed on the graphics screen. The intensities in the Fourier transform image indicate dominating pattern repetitions in the original image. The intensity of the center pixel in the Fourier transform is proportional to the average brightness of the original image (at pattern frequency equal to zero). The intensities of Fourier pixels further from the center indicate how "dominant" denser pattern repetitions are in the original image.

Specifically, a high Fourier intensity at a distance of *x* pixels to the right of the center of the Fourier image, indicates that there is a regular horizontal pattern in the original image with a repetition period of *512/(2x)* pixels (assuming that the original image has 512 pixels from left to right). The same holds for analysis of vertical patterns.

8-Band Fourier Transform: This operation is similar to the "Fourier Transform" option, above, except here the image is segmented into 8 subwindows, each one spanning the full width of the image (from left to right) but only 1/8th of the total image height. A Fourier transform is taken of each of the subwindows. This operation was used in early experiments to determine the frequency or spacing of weld ripples, but the vertical/horizontal integration procedure was later developed and found to be more reliable.

16-Band Fourier Transform: Similar to the "8-Band Fourier Transform", except here the image is segmented into 16 horizontal subwindows.

Edge:

Roberts:

The edge operations analyze the displayed image and replace all pixel intensities with new values that are indicative of brightness variations in their neighborhoods. The Roberts operator is essentially the gradient operator. Thus, each pixel intensity is replaced by the original image gradient at that location. The resulting image highlights edges and sharp lines, which characterize many weld defects.

Sobel:

The Sobel operator is an indicator of the gradients in the image, but the gradient is computed from a 3-by-3 neighborhood around each pixel, rather than the 2-by-2 which is used by the Roberts operator. The result is an edge finding operator that is less susceptible to individual noise pixels than the Roberts operator.

Prewitt:

The Prewitt operator is similar to the Sobel operator in that the local gradient is estimated from a 3-by-3 matrix. To improve noise performance further, however, it assigns different weights to the pixels in the matrix.

Kirsch:

Unlike the edge operators listed above, which all estimate the gradient by calculating the horizontal derivative and the vertical derivative and then combining both, the Kirsch approach also determines the two diagonal derivatives before the total gradient is calculated. The local matrix used for this calculations is 3-by-3 pixels.

Help:

Commands:

Option for on-line help.

About Imagine:

Displays the name of the software and its producer.

This concludes description of the prototype software system developed for this work. In short, in includes many of the tools that provide the foundation for visual inspection for quality control. This prototype serves as a template for a commercial quality inspection system that is presently under consideration.

CHAPTER VII

CONCLUSIONS AND SUGGESTIONS FOR COMMERCIALIZATION

The work described in this report has been aimed at exploring means for using readily available computer technology in the process of quality inspection and evaluation. The IBM-PC computer architecture was selected for the purpose of facilitating commercialization for as wide a range of applications as possible. To demonstrate the basic principles and potentials of computerized inspection, off-line quality control for arc welding was selected as a representative application. Visual inspection was emphasized, as it requires relatively modest sensory hardware. Furthermore, visual inspection is widely applicable and is in itself an effective inspection technique.

The results of this work illustrate that visual quality inspection may be aided and to some extent automated with relatively modest computer resources. A prototype software system was developed for the work, which is delivered to NASA at the conclusion of this contract. This software may be executed with any compatible computer and image processing hardware. Furthermore, a number of algorithms and related techniques were developed to aid in evaluating weld quality.

The next step in this work will be the development of a commercial quality inspection system, based on the prototype system developed under this contract. The commercial system may be marketed as an integrated package, as well as separate hardware and software components. Demonstrations and discussions of the prototype system have been conducted with potential commercial customers and their comments will be used in guiding the development of the commercial system.

BIBLIOGRAPHY

1. Anderson, R.C., "Inspection of Metals: Visual Examination", American Society for Metals, Metals Park, Ohio, 1983.

2. *Radiography In Modern Industry*, 3rd ed., Rochester, New York: Eastman Kodak Company, 1969.

3. *ASTM E709, Standard Recommended Practice for Magnetic Particle Examination,* American Society for Testing and Materials, Philadelphia, PA.

4. *ASTM E165, Standard Recommended Practice for Liquid Penetrant Inspection*, American Society for Testing and Materials, Philadelphia, PA.

5. *ASTM E164, Standard Practice for Ultrasonic Contact Examination of Weldments,* American Society for Testing and Materials, Philadelphia, PA.

6. *ASTM E749, Standard Practice for Acoustic Emission Monitoring During Continuous Welding,* American Society for Testing and Materials, Philadelphia, PA.

7. Liptai, R.G., Harris, D.O., and Tatro, C.A., "Acoustic Emission, A Symposium", presented at December (1971) Committee Week, ASTM Special Technical Publication 505.

8. Spanner, J.C., "Acoustic Emission Techniques and Applications", Evanston, Illinois, Intex Publishing Co.

9. CLIPS - C Language Integrated Production System, Users Manual, Distributed by COSMIC, 16511 Space Center Boulevard, Houston, Texas 77058.

10. Andersen, K., Barnett, R.J., Springfield, J.F., and Cook, G.E. "Welding Technology: VPPAW Modeling And Control Using Artificial Neural Networks." SBIR Phase I Final Report, NASA Contract NAS8-38918, Mid-South Engineering, Inc., Nashville, Tennessee, August, 1991.

11. Rosenblatt, F., "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain," Psych. Rev. 65, 386, 1958.

12. Rosenblatt, F., "Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms," Spartan Books, 1962.

13. Minsky, M.A. and Papert, S.A., "Perceptrons - Expanded Edition," MIT Press, Cambridge, MA, 1988 (revised and reprinted).

14. Werbos, P., "Beyond Regression," Ph.D. Dissertation, Harvard University, August, 1974.

15. Parker, D.B., "Learning Logic," Technical Report TR-47, Center for Computational Research in Economics and Management Science, MIT, April, 1985.

16. Rumelhart, D.E., Hinton, G.E., and Williams, R.J., "Learning Internal Representations by Error Propagation," in *Parallel Distributed Processing*, (D.E. Rumelhart and J.L. McClelland, eds.), ch. 8, MIT Press, Cambridge, MA, 1986.

17. DARPA Neural Network Study, AFCEA International Press, Fairfax, Virginia, November, 1988.

18. Andersen, K., Cook, G.E., Ramaswamy, K., Karsai, G., "Artificial Neural Networks Applied to Arc Welding Process Modeling and Control", *Proc. IEEE Industry Applications Society (IAS) 1989 Annual Meeting*, San Diego, California, October 1-5, 1989, pp. 2327 - 2331

19. Andersen, K., Barnett, R.J., Springfield, J.F., Cook, G.E., Strauss, A.M., and Bjorgvinsson, J.B. "Intelligent Gas Tungsten Arc Welding Control." SBIR Phase II Final Report, NASA Contract NAS8-37401, Mid-South Engineering, Inc. / Vanderbilt University, Nashville, Tennessee, February, 1991.

20. Andersen, K., Cook, G.E., Springfield, J.F., and Barnett, R.J., "Applications Of Artificial Neural Networks For Arc Welding," Eds: Dagli, C.H., Kumara, S.R.T. and Shin, Y.C., Proceedings of Artificial Neural Networks in Engineering (ANNIE '91), ASME Press Series on International Advances in Design Productivity, St. Louis, Missouri, Nov., 1991.

21. NeuralWare, Inc., Penn Center West, Building IV, Suite 227, Pittsburgh, PA 15276. (412) 787-8222.

APPENDIX A

Artificial Neural Networks

Historical Overview

Artificial neural networks are implemented in various configurations and some are better suited for certain applications than others.

One of the earliest accomplishments in neural network implementation was the "perceptron" which Rosenblatt designed [11,12]. The perceptron was a primitive pattern classifier, designed to recognize optical patterns. A grid of 400 photocells (simulating light-sensitive neurons) sensed an illuminated pattern and each photocell produced an output signal according to the light intensity it received. An array of "associator units" was used to select some of the photocell signals for further processing, while others were rejected. The selected signals were transmitted to a summing device, and the final output of the perceptron was thus the sum of the signals from the associator units. The output of the perceptron was used as an indicator of whether the applied optical pattern belonged to the class of patterns that the perceptron has been trained to recognize, or not. The associator units, which determined which photocell signals were of importance, essentially determined the class of patterns recognized by the perceptron. The perceptron had certain limitations, the most severe one being that it could not learn to classify any arbitrary patterns. This lack of generality was overcome later by the introduction of multiple-network structures and adding a nonlinearity transfer function to the summing node.

Minsky and Papert analyzed the basic perceptron further [13] and demonstrated that, for example, that perceptrons could not distinguish connected pattern figures from unconnected ones. Furthermore, they showed that the basic perceptron can not compute functions that are not linearly separable, such as the Exclusive-OR Boolean function. Multiple-layer perceptrons, using nonlinear transfer functions, were introduced to overcome the limitations of the basic perceptron. Although these structures were shown to be more powerful than the original perceptron, they did not gain general interest until the backpropagation algorithm was devised [14,15,16]. This algorithm provided means for systematically adjusting the various weights of the multi-layer network until it had been adequately trained.

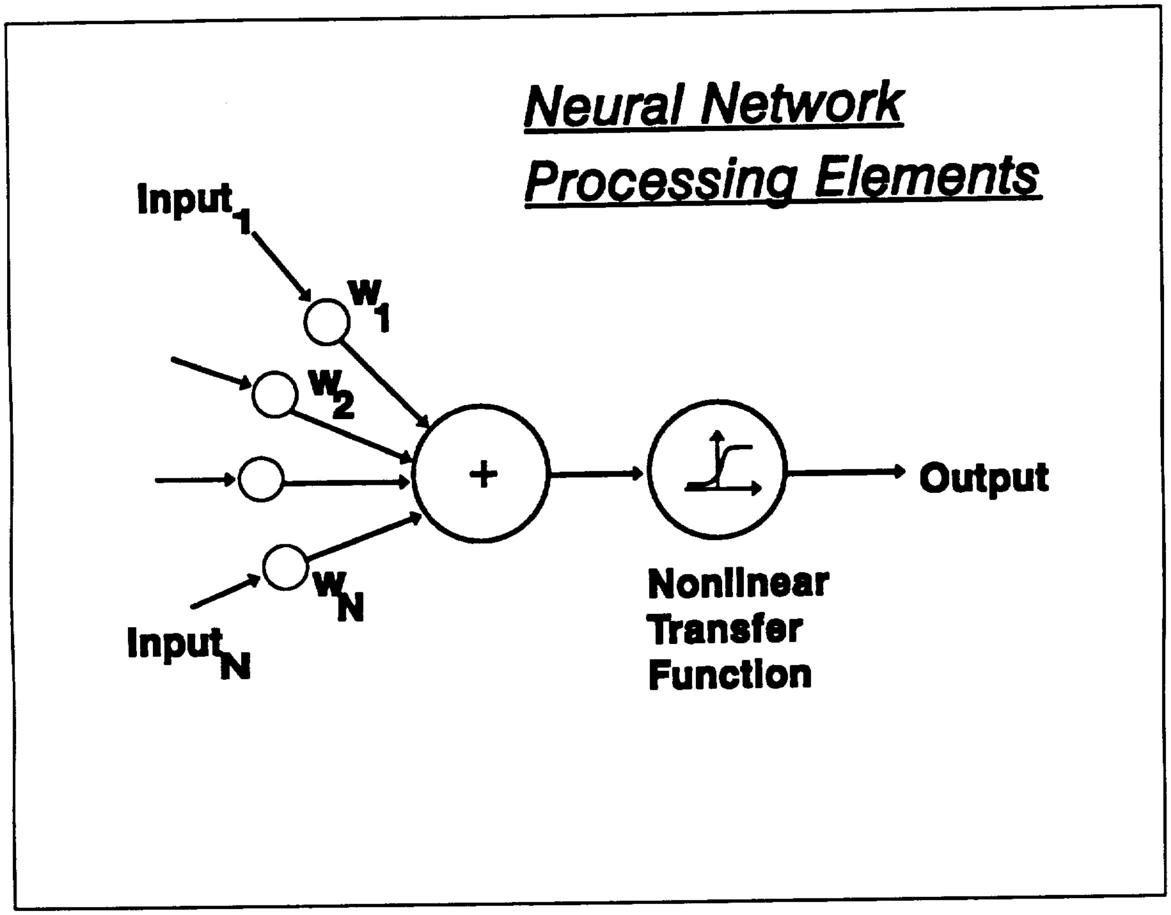


Figure 50. The processing element of a neural network.

The multi-layer networks consist of a number of *processing elements*, which to a certain degree resemble the connected neurons in biological systems (refer to Figure 50). A processing element accepts one or more signals, which may be produced by other processing elements or applied externally (e.g., provided by a process sensor). The various signals are individually amplified, or weighted, and then summed together within the processing element. The resulting sum is applied to a specified *transfer function*, and the function value becomes the output of the processing element.

The Backpropagation Network

A neural network and its adaptation procedure using the backpropagation algorithm is best illustrated by an example.

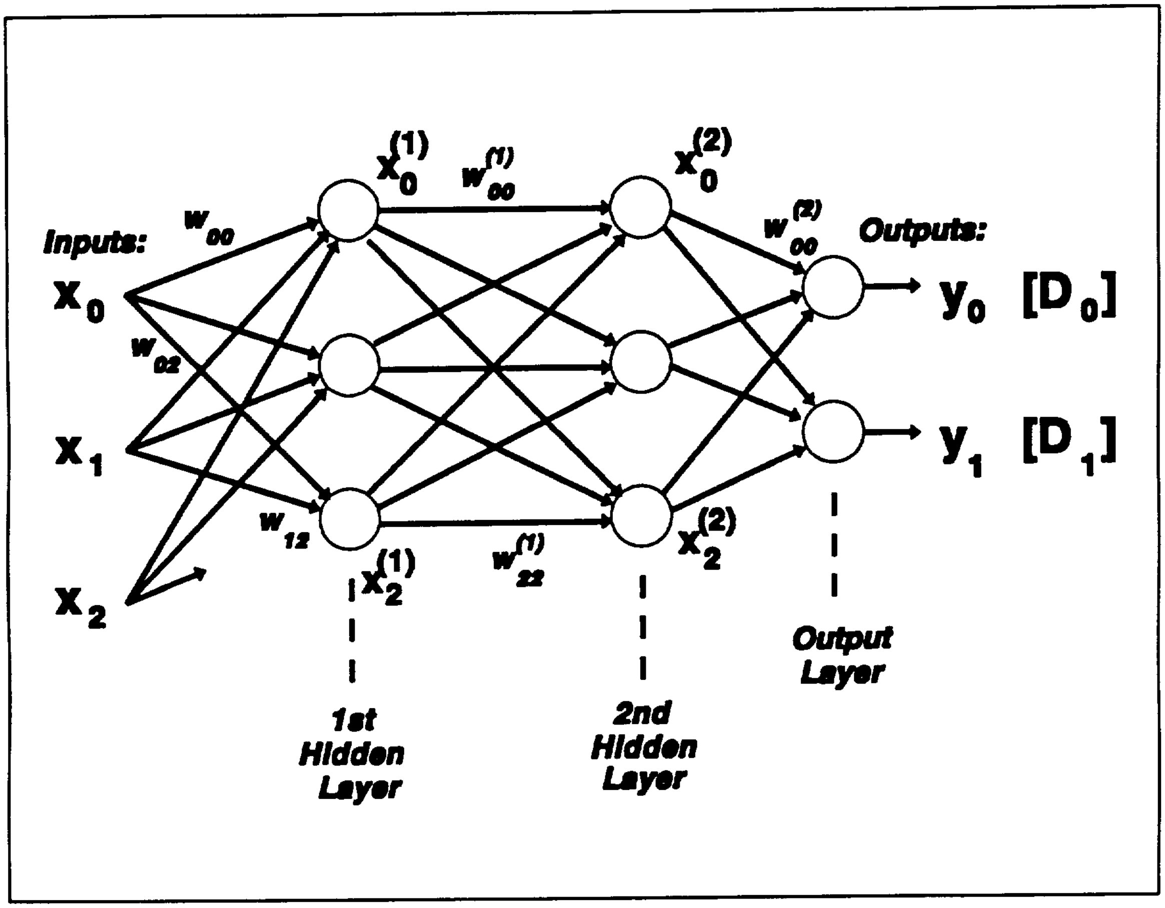


Figure 51. A backpropagation neural network.

Figure 51 shows a small neural network consisting of 8 *nodes*, arranged in two *hidden layers* of 3 nodes each and one *output layer* of 2 nodes. Each node i in the first hidden layer produces a single numeric output which we denote as $x_i^{(1)}$. Similarly the nodes of the second hidden layer are labelled $x_0^{(2)}$ through $x_2^{(2)}$. The 3 inputs and 2 outputs of the network are x_0 through x_2 and y_0 through y_1 respectively. Each node accepts numeric data through a number of *input links*, each of which multiplies the input data with a *weight* factor. The weight factor associated with the link from $x_i^{(1)}$ to the node producing $x_j^{(2)}$ is annotated as $w_{ij}^{(1)}$ and a similar convention holds for the links between other layers. Each node calculates its output by summing its weighted inputs and using the result s as the argument of a nonlinear function associated with the node. For our application this function is the same for all nodes:

$$f(s) = [1 + \exp[-(s-c)]]^{-1} \tag{11}$$

where *s* is the sum of the node inputs and *c* is an internal offset value. Clearly the node output will be confined to the range $0 < f(s) < 1$. Because the limiting values, 0 and 1, will only be approached as s approaches +/- infinity all input and output data are scaled so that they are confined to a subinterval of [0...1]. A practical region for the data is chosen to be [0.1...0.9]. In this case each input or output parameter p is normalized as p_n before being applied to the neural network according to:

$$p_n = [(0.9-0.1)/(p_{max}-p_{min})](p-p_{min})+0.1 \tag{12}$$

where p_{max} and p_{min} are the maximum and minimum values, respectively, of data parameter p. The network starts calculating its output values by passing the weighted inputs to the nodes in the first layer. The resulting node outputs of that layer are passed on, through a new set of weights, to the second layer, and so on until the nodes of the output layer compute the final outputs.

Before practical application, the network has to be *trained* to perform the mapping of the three input parameters to the two output parameters. This is done by repeatedly applying training data to its inputs, calculating the corresponding outputs by the network, comparing them to the desired outputs, and altering the internal parameters of the network for the next round. The training starts by assigning small random values to all weights (w_{ij}) and node offsets (c_j) in the network. The first three input data values are presented to the network which in turn calculates the two output values. Because the initial weights and node offsets are random these values will generally be quite different from the desired output values, D_0 and D_1. Therefore the differences between the desired and calculated outputs have to be utilized to dictate improved network values, tuning each weight and offset parameter through back propagation. The weights preceding each output node are updated according to

$$w_{ij}(t+1) = w_{ij}(t) + nd_jx_i^{(2)} \tag{13}$$

where *n* is a *correction gain* and d_j is the *correction factor*

$$d_j = y_j\,(1-y_j)\,(d_j-y_j) \tag{14}$$

Clearly, each weight will be increased if the calculated output from its node is less than the desired value, and vice versa. The correction factors used to update weights preceding hidden layer nodes are updated according to

$$d_j \;=\; x_j \cdot (1-x_j) \sum_k (d_k \cdot w_{jk}) \tag{15}$$

where the k applies to the node layer succeeding the one currently being updated. The offset parameter c of each node is treated as an additional weight factor and updated in the same manner.

The weights and offsets of the neural network are recalculated during the backpropagation as outlined above. Then the network repeats calculation of output values based on the same input data, compares them to the desired output values, and readjusts the network parameters through yet another backpropagation phase. This cycle is repeated until the calculated outputs have converged sufficiently close to the desired outputs or an iteration limit has been reached. Once the neural network has been tuned to the first set of input/output data, additional data sets can be used for further training in the same way. To ensure concurrent network adaptation to all sets of data the entire training process may be repeated until all data transformations are adequately modeled by the network. This requires, of course, that all the data sets were obtained from the same physical process and therefore the underlying input/output transformation is consistent.

As noted above, the training iteration process may be terminated either by a convergence limit or simply by limiting the total number of iterations. In the former case we use an error measure *e* defined as following:

$$e \;=\; \max_{k=1..K} \;\{\, \sum_{m=0}^{M-1} (d_{k,m}-y_{k,m})^2 \}\tag{16}$$

where K is the number of input/output data sets used for training, M is the number of network output parameters in each data set, and $(d_{k,m}y_{k,m})$ is the error in the network calculation of parameter m in data set k. The error measure, e, changes after each round of network weight adjustments. In the long run e decreases as the network is refined by training iterations. Using this indicator one can program the network to terminate the iterative tuning process as soon as e reaches some threshold value, e_0. Alternatively, a given network may not be able to reduce the error measure down to the specified e_0. In that case the iterations may be terminated by simply specifying a maximum number for them.

The *training mode*, as described above, is a precondition for actually applying the neural network in the *application mode*. In this mode entirely new input data is presented to the network which, in turn, predicts new outputs based on the transfer characteristics learned during the training. If this new data is obtained from the same local region of operation of the process as during the training phase, data from the input/output relations should be governed by the same underlying process and the neural network should perform adequately. The neural network is not updated in the application mode.

The above discussion should suffice to give an overview of neural networks and how they work. The reader is referred to the bibliography in this report for further information on the mathematics and theory of artificial neural networks.

Neural Network Applications

Artificial neural networks have been used in a number of applications, ranging from radar signal processing to financial analysis to robotics.

One of the earliest application of neural networks is pattern classification. Several military applications are currently under development, where neural networks are implemented to recognize targets in real-time. It is believed that a proximity fuze for a smart weapon can be realized by a multi-layer neural network, which in turn, can be implemented on a single silicon chip [17].

Process modeling and control, is one are in which neural networks have proven applicable. The application of neural networks for modeling and control of the arc welding processes has been pioneered by the authors of this report and the results have been published in [18,19,20]

Neural networks are likely to appear in robotic manipulators for trajectory control. Simulations have proven to be successful and it appears to be only a matter time before specific hardware becomes available for this purpose.

Finally, signal processing is one are where neural networks have been tried with some success. Because neural networks provide an adaptive, nonlinear capability, they appear to offer a promising alternative to more traditional efforts in this area.

Neural Network Hardware and Speed Considerations

A number of hardware accelerators and dedicated neural network computers are presently in the design phase or prototype production stage. Furthermore, a number of neural network software vendors have optimized their code for generic numerical accelerator boards or chips. The use of such dedicated hardware components may be desirable if substantial amount of re-training is required in a production system.

NeuralWare, Inc. [21] developed the software that was used for most of the neural network simulations presented in this work. This vendor offers a generic numerical accelerator board (ALACRON) which increases the execution rate of the neural networks by a factor of approximately an order of magnitude. Using this board, typical execution times for a 15-node network with 2 inputs and 4 outputs is less than 1 millisecond.

More experimental neural network hardware includes the Network Emulation Processor, developed by Claude Cruz-Young, who was previously at IBM Research Laboratories. This is a medium-scale VLSI-based virtual processor. It is currently used as an emulator for the Parallel Associative Network (PAN) workstation with an IBM PC.

ANZA and ANZA Plus are two boards designed specifically for neural network simulation, and developed at the Hecht-Nielsen Neuro-Computing Corporation. Both boards are based on the Motorola 68xxx series processors and co-processors. The ANZA board can achieve up to 45,000 inter-connects per second, while the ANZA Plus reaches as high as 1,500,000 inter-connects per second.

Neural networks are, although on a small scale so far, implemented on dedicated silicon chips. AT&T has been developing a neural computing chip that has 256 neuron-transistors and more than 100,000 synapse-resistors. These resistors are "burned in" at the time of manufacturing, so training and simulation of the application must be completed before the chips are ordered. AT&T is currently working on a re-programmable chip.

It is important to distinguish between the training speed and execution speed of neural networks. The training speed of neural networks is generally very low. During the training process the internal parameters of the network are iteratively adjusted until the network training has been optimized. For the backpropagation network the network weights are the adjustable parameters. The training is usually carried out off-line and optimized before the network is finally applied in the application. The task of weld profile analysis with neural networks, which was demonstrated for this work, was found to be executed in a fraction of a second on relatively modest computer systems, which is adequate for most inspection applications.

APPENDIX B

Source Code Listing for the Quality Analysis System

```
# ----- File MK_IMAGE -----

CFLAGS = -c -AL -G2sw -Owx -W3 -FPi87
CFLAGSC = -c -AL -G2s -Ox -W3 -FPi87
#CFLAGS = -c -AL -G2sw -Zi -Od -W3 -FPi87
#CFLAGSC = -c -AL -G2s -Zi -Od -W3 -FPi87

all: image.exe

image.obj: image.c dialog.h menu.h
    cl $(CFLAGS) image.c

process.obj: process.c
    cl $(CFLAGSC) process.c

edge.obj: edge.c
    cl $(CFLAGSC) edge.c

integ.obj: integ.c
    cl $(CFLAGSC) integ.c

hist.obj: hist.c dialog.h
    cl $(CFLAGS) hist.c

fourn.obj: fourn.c
    cl $(CFLAGSC) fourn.c

texture.obj: texture.c
    cl $(CFLAGSC) texture.c

image.res: image.rc dialog.h image.ico menu.h dialog.dlg
    rc -r image

image.exe: image.obj process.obj edge.obj hist.obj mapn.obj copy.obj image.def integ.obj fourn.obj
texture.obj
    link /co /align:16 @image.rsp
    rc image.res

image.exe: image.res
    rc image.res
```

```
/* ----- File DIALOG.H ----- */

#define IDB_AUTO         1013
#define IDB_SAVE         1012
#define IDT_INT          1011
#define IDE_INT          1010
#define IDS_INT          1009
#define IDB_CANCEL       2
#define IDB_OK           1
#define ID_RESOURCE      999
#define ID_HISTO         120
#define IDD_ABOUT        1001
#define IDD_SCROLL       1005
#define IDD_BUF          1002
#define IDD_HISTO        1003
#define IDD_INTEGRATE    1004
#define IDD_PATH         110
#define IDD_FILEEDIT     111
#define IDD_DIRLIST      112
#define IDD_FILELIST     113
#define IDD_HEAD         114
#define IDD_SCROLLHEAD   130
#define IDD_SCROLLBAR    131
#define IDD_SCROLLNUM    132
#define IDC_INT          140
#define IDC_SAT          141
#define IDC_HUE          142
#define IDC_OVL          143
#define IDC_CLR          144
#define IDW_HWINI        150
#define IDW_HWINH        151
#define IDW_HWINS        152
#define IDB_16           153
#define IDB_32           154
#define IDB_64           155
#define IDB_128          156
#define IDB_256          157
#define IDB_HUE          158
#define IDB_SAT          159
#define IDB_INT          160
#define IDD_FILEIO       1000
#define IDW_BAR          1006
#define IDB_UPDATE       1008
```

```
/* ----- File EDGE.H ----- */

int sobel_mask[8][3][3] ={
                         {{1,2,1},{0,0,0},{-1,-2,-1}},
                         {{2,1,0},{1,0,-1},{0,-1,-2}},
                         {{1,0,-1},{2,0,-2},{1,0,-1}},
                         {{0,-1,-2},{1,0,-1},{2,1,0}},
                         {{-1,-2,-1},{0,0,0},{1,2,1}},
                         {{-2,-1,0},{-1,0,1},{0,1,2}},
                         {{-1,0,1},{-2,0,2},{-1,0,1}},
                         {{0,1,2},{-1,0,1},{-2,-1,0}}
                         };

int prewitt_mask[8][3][3] = {
                         {{1,1,1},{0,0,0},{-1,-1,-1}},
                         {{1,1,0},{1,0,-1},{0,-1,-1}},
                         {{1,0,-1},{1,0,-1},{1,0,-1}},
                         {{0,-1,-1},{1,0,-1},{1,1,0}},
                         {{-1,-1,-1},{0,0,0},{1,1,1}},
                         {{-1,-1,0},{-1,0,1},{0,1,1}},
                         {{-1,0,1},{-1,0,1},{-1,0,1}},
                         {{0,1,1},{-1,0,1},{-1,-1,0}}
                          };

int kirsch_mask[8][3][3] = {
                         {{5,5,5},{-3,0,-3},{-3,-3,-3}},
                         {{5,5,-3},{5,0,-3},{-3,-3,-3}},
                         {{5,-3,-3},{5,0,-3},{5,-3,-3}},
                         {{-3,-3,-3},{5,0,-3},{5,5,-3}},
                         {{-3,-3,-3},{-3,0,-3},{5,5,5}},
                         {{-3,-3,-3},{-3,0,5},{-3,5,5}},
                         {{-3,-3,5},{-3,0,5},{-3,-3,5}},
                         {{-3,5,5},{-3,0,5},{-3,-3,-3}}
                          };
```

```
/* ----- File MENU.H ----- */

#define IDM_FILE 10
#define IDM_ACQUIRE 11
#define IDM_ACQUIRERGB 12
#define IDM_OPEN 13
#define IDM_SAVEAS 14
#define IDM_BUFFERS 15
#define IDM_EXIT 16

#define IDM_PROCESS 20
#define IDM_BLOWUP 21
#define IDM_ZOOM 22
#define IDM_ENLARGE 23
#define IDM_ENHRED 24
#define IDM_EQUALIZE 25
#define IDM_THRESHOLD 26
#define IDM_AVERAGE 27
#define IDM_INTEGRATE 28
#define IDM_HOR 29
#define IDM_VER 30
#define IDM_HISTO 31
#define IDM_INT 32
#define IDM_SAT 33
#define IDM_HUE 34
#define IDM_INVERSE 35

#define IDM_EDGE 40
#define IDM_ROBERTS 41
#define IDM_SOBEL 42
#define IDM_PREWITT 43
#define IDM_KIRSCH 44

#define IDM_TEXTURE 50
#define IDM_ONEITER 51
#define IDM_MULTITHRESH 52
#define IDM_DOITALL 53
#define IDM_FOURIER 54
#define IDM_BAND_8 55
#define IDM_BAND_16 56

#define IDM_HELP 60
#define IDM_HELP_COMMANDS 61
#define IDM_ABOUT 62

#define ID_FRAME 99
```

```
/* ----- File COPY.C ----- */

void copy(unsigned char huge *,unsigned char huge *);

void copy(ipic,opic)
unsigned char huge *ipic;
unsigned char huge *opic;
{
unsigned int i;
for (i=0;i<4;i++) {
    opic += 65536L;
    ipic += 65536L;
    *opic = *ipic;
    }
}
```

```
/* ----- File EDGE.C ----- */

#include <math.h>
#include "util.h"
#include "edge.h"

#define HEIGHT 512
#define WIDTH 512
extern unsigned char huge ipic[HEIGHT][WIDTH];
extern unsigned char huge opic[HEIGHT][WIDTH];

extern int ha;
void roberts_main(void);
void sobel_main(void);
void prewitt_main(void);
void kirsch_main(void);
unsigned char mask_max(int (*)[3][3],int,int);
void maskit(int mask[8][3][3]);

void roberts_main(void) {
int i,j,d1,d2,p;
unsigned char huge *tpic;
tpic = (unsigned char huge *) ipic;
au_readscr(ha,ipic,0);
for (i=0; i<HEIGHT-1; i++,tpic++) {
        for (j=0; j<WIDTH-1; j++,tpic++) {
                d1 = (int) *(tpic+1) - (int) *tpic;
                d2 = (int) *(tpic+WIDTH) - (int) *tpic;
                p = (int) (.707 * sqrt((double) d1*d1+d2*d2));
                *tpic = (unsigned char) p;
                }
        }
au_writescr(ha,ipic,0);
}

void maskit(mask)
int mask[8][3][3];
{
int i,j;
int a,c,t;
int (*m)[3];
unsigned char huge *topic = (unsigned char huge *) (opic[1]);
au_readscr(ha,ipic,0);
for (i=1; i<HEIGHT-1; i++,topic+=2) {
    for (j=1; j<WIDTH-1; j++,topic++) {
        for (t=0,a=0;a<4;a++) {
            m = mask[a];
            c = ((int)ipic[i-1][j-1])*m[0][0];
            c += ((int)ipic[i][j-1])*m[0][1];
            c += ((int)ipic[i+1][j-1])*m[0][2];
            c += ((int)ipic[i-1][j])*m[1][0];
            c += ((int)ipic[i][j])*m[1][1];
            c += ((int)ipic[i+1][j])*m[1][2];
            c += ((int)ipic[i-1][j+1])*m[2][0];
            c += ((int)ipic[i][j+1])*m[2][1];
            c += ((int)ipic[i+1][j+1])*m[2][2];
            c = abs(c)/4;
            if (c>t) t=c;
            }
        *topic = (unsigned char) t;
        }
    }
au_writescr(ha,opic,0);
}

void sobel_main(void) {
maskit(sobel_mask);
}

void prewitt_main(void) {
maskit(prewitt_mask);
}

void kirsch_main(void) {
maskit(kirsch_mask);
}
```

```
/* ----- File FOURN.C ----- */

#define INCL_WIN
#define INCL_DOS
#include <os2.h>
#include <math.h>
#include <util.h>

#define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr
void fourn(float huge *,long *,int,int);
extern HWND hwndFrame;
extern unsigned char huge tpic[512][512];
extern unsigned char huge opic[512][512];
extern int ha;

void fft(void)
{
float huge *arr;
float huge *tarr;
float tf1,tf2,mx=(float)0.;
SEL selHuge;
static long dims[2]={512L,512L};
long li,lj;
unsigned char huge *tp = (unsigned char huge *)tpic;
if (DosAllocHuge(32,0,&selHuge,0,SEG_NONSHARED)) {  ; /* allocate 2Meg of memory */
    DosBeep(2000,1000);
    WinMessageBox(HWND_DESKTOP,hwndFrame,"Could not allocate memory",NULL,0,MB_OK |
MB_ICONEXCLAMATION);
    return;
    }
tarr = arr = (float huge *) MAKEP(selHuge,0);
DosBeep(8000,300);
au_readscr(ha,tpic,0); /* read intensity data */
for (li=0L;li<262144L;li++) {
    *tarr = (float) *tp;
    tarr++;
    *tarr = (float)0.;
    tp++;
    tarr++;
    }
tarr = arr;
tp = (unsigned char huge *)tpic;
fourn(arr,dims,2,1);
for (li=0L;li<262144L;li++,tarr+=2) {
    tf1 = *tarr;
    tf2 = *(tarr+1);
    *tarr = (float) log(sqrt(tf1*tf1+tf2*tf2));
    if (*tarr > mx) mx = *tarr;
    }
tarr = arr;
for (li=0L;li<262144L;li++,tarr+=2,tp++) {
    tf1 = *tarr/mx*(float)255.;
    if (tf1 < (float)0.) tf1 = (float)0.;
    *tp = (unsigned char) tf1;
    }
for (li=0L;li<256L;li++)
    for (lj=0L;lj<256L;lj++) {
        opic[li+256L][lj+256L] = tpic[li][lj];
        opic[li][lj] = tpic[li+256L][lj+256L];
        opic[li+256L][lj] = tpic[li][lj+256L];
        opic[li][lj+256L] = tpic[li+256L][lj];
        }
au_writescr(ha,opic,0); /* write intensity data */
DosFreeSeg(selHuge);
}

void bfft(bands)
int bands;
{
float huge *arr;
float huge *tarr;
float tf1,tf2,mx=(float)0.;
SEL selHuge;
static long dims[2]={64L,512L};
long li,lj,lk;
unsigned char huge *tp = (unsigned char huge *)tpic;
dims[0] = 512L/(long)bands;
if (DosAllocHuge(32,0,&selHuge,0,SEG_NONSHARED)) {  ; /* allocate 2Meg of memory */
    DosBeep(2000,1000);
    WinMessageBox(HWND_DESKTOP,hwndFrame,"Could not allocate memory",NULL,0,MB_OK |
MB_ICONEXCLAMATION);
    return;
    }
tarr = arr = (float huge *) MAKEP(selHuge,0);
DosBeep(8000,300);
au_readscr(ha,tpic,0); /* read intensity data */
for (li=0L;li<262144L;li++) {
    *tarr = (float) *tp;
    tarr++;
    *tarr = (float)0.;
    tp++;
    tarr++;
    }
tp = (unsigned char huge *)tpic;
for (li=0L;li<512/dims[0];li++) {
    tarr = &arr[li*dims[0]*dims[1]*2];
    fourn(tarr,dims,2,1);
    }
tarr = arr;
for (li=0L;li<262144L;li++,tarr+=2) {
    tf1 = *tarr;
    tf2 = *(tarr+1);
    *tarr = (float) sqrt(tf1*tf1+tf2*tf2);
    if (*tarr > (float)1.)
        *tarr = (float)log(*tarr);
    else
        *tarr = (float)0.;
    if (*tarr > mx) mx = *tarr;
    }
tarr = arr;
for (li=0L;li<262144L;li++,tarr+=2,tp++) {
    tf1 = *tarr/mx*(float)255.;
    *tp = (unsigned char) tf1;
    }
for (lk=0L;lk<512/dims[0];lk++) {
    for (li=0L;li<dims[0]/2;li++) {
        for (lj=0L;lj<256L;lj++) {
            opic[lk*dims[0]+li][lj] = tpic[lk*dims[0]+dims[0]/2+li][lj+256];
            opic[lk*dims[0]+dims[0]/2+li][lj+256] = tpic[lk*dims[0]+li][lj];
            opic[lk*dims[0]+li][lj+256] = tpic[lk*dims[0]+dims[0]/2+li][lj];
            opic[lk*dims[0]+dims[0]/2+li][lj] = tpic[lk*dims[0]+li][lj+256];
            }
        }
    }
au_writescr(ha,opic,0); /* write intensity data */
DosFreeSeg(selHuge);
}


void fourn(data,nn,ndim,isign)
float huge *data;
int ndim,isign;
long *nn;
{
        long i1,i2,i3,i2rev,i3rev,ip1,ip2,ip3,ifp1,ifp2;
        long ibit,k1,k2,n,nprev,nrem,ntot;
            int idim;
        float tempi,tempr;
        double theta,wi,wpi,wpr,wr,wtemp;

        ntot=1L;
        for (idim=0;idim<ndim;idim++)
                ntot *= nn[idim];
        nprev=1L;
        for (idim=ndim-1;idim>=0;idim--) {
                        DosBeep(1000,500);
                n=nn[idim];
                nrem=ntot/(n*nprev);
                ip1=nprev << 1;
                ip2=ip1*n;
                ip3=ip2*nrem;
                i2rev=1;
                for (i2=1;i2<=ip2;i2+=ip1) {
                        if (i2 < i2rev) {
                                for (i1=i2;i1<=i2+ip1-2;i1+=2) {
                                        for (i3=i1;i3<=ip3;i3+=ip2) {
                                                i3rev=i2rev+i3-i2;
                                                SWAP(data[i3-1],data[i3rev-1]);
                                                SWAP(data[i3],data[i3rev]);
                                        }
                                }
            }
            ibit=ip2 >> 1;
            while (ibit >= ip1 && i2rev > ibit) {
                i2rev -= ibit;
                ibit >>= 1;
            }
            i2rev += ibit;
        }
        ifp1=ip1;
        while (ifp1 < ip2) {
            ifp2=ifp1 << 1;
            theta=isign*6.28318530717959/(ifp2/ip1);
            wtemp=sin(0.5*theta);
            wpr = -2.0*wtemp*wtemp;
            wpi=sin(theta);
            wr=1.0;
            wi=0.0;
            for (i3=1;i3<=ifp1;i3+=ip1) {
                for (i1=i3;i1<=i3+ip1-2;i1+=2) {
                    for (i2=i1;i2<=ip3;i2+=ifp2) {
                        k1=i2;
                        k2=k1+ifp1;
                        tempr=wr*data[k2-1]-wi*data[k2];
                        tempi=wr*data[k2]+wi*data[k2-1];
                        data[k2-1]=data[k1-1]-tempr;
                        data[k2]=data[k1]-tempi;
                        data[k1-1] += tempr;
                        data[k1] += tempi;
                    }
                }
                wr=(wtemp=wr)*wpr-wi*wpi+wr;
                wi=wi*wpr+wtemp*wpi+wi;
            }
            ifp1=ifp2;
        }
        nprev *= n;
    }
}

#undef SWAP
```

```
/* ----- File HIST.C ----- */

#define INCL_WIN
#define INCL_DOSPROCESS
#include <os2.h>
#include <stdio.h>
#include "dialog.h"

extern HWND hwndFrame;
static char buf[80];

MRESULT EXPENTRY HistoWndProc(HWND hwnd, USHORT msg,MPARAM mp1,MPARAM mp2)
{
HPS hps;
USHORT i;
ULONG width,height;
float incx,fx;
POINTL pt;
SWP swp;
unsigned int num_hist_levels;
float *hist;
switch(msg) {
    case WM_PAINT:
        num_hist_levels = WinQueryWindowUShort(hwnd,8);
        hist = (float *) WinQueryWindowPtr(hwnd,0);
        hps = WinBeginPaint(hwnd,NULL,NULL);
        GpiErase(hps);
        GpiSetPattern(hps,PATSYM_DIAG1);
        WinQueryWindowPos(hwnd,&swp);
        width = (ULONG) swp.cx;
        height = (ULONG) swp.cy;
        incx = (float) width / (float) num_hist_levels;
        pt.x = 0L; pt.y=0L;
        GpiMove(hps,&pt);
        pt.x = width-1L; pt.y = height-1L;
        GpiBox(hps,DRO_OUTLINE,&pt,0L,0L);
        fx = (float) 0.;
        pt.x = 0L;
        for (i=0;i<num_hist_levels;i++) {
            pt.y = 0L;
            GpiMove(hps,&pt);
            fx += incx;
            pt.x = (long) fx;
            pt.y += (long) ( (float) height * hist[i]);
            GpiBox(hps,DRO_OUTLINE,&pt,0L,0L);
            DosBeep( (USHORT) ((float)height*(float)10.*hist[i]),10);
            }
        WinEndPaint(hps);
        return 0;
    case WM_ERASEBACKGROUND:
        return 1;
    }
return WinDefWindowProc(hwnd,msg,mp1,mp2);
}


MRESULT EXPENTRY IntWndProc(HWND hwnd, USHORT msg,MPARAM mp1,MPARAM mp2)
{
HPS hps;
USHORT i;
ULONG width,height;
float incx,fx;
POINTL pt;
SWP swp;
float *inte;
switch(msg) {
    case WM_PAINT:
        inte = (float *) WinQueryWindowPtr(hwnd,0);
        hps = WinBeginPaint(hwnd,NULL,NULL);
        GpiErase(hps);
        GpiSetPattern(hps,PATSYM_DIAG1);
        WinQueryWindowPos(hwnd,&swp);
        width = (ULONG) swp.cx;
        height = (ULONG) swp.cy;
        incx = (float) width / (float) 256;
        pt.x = 0L; pt.y=0L;
        GpiMove(hps,&pt);
        pt.x = width-1L; pt.y = height-1L;
        GpiBox(hps,DRO_OUTLINE,&pt,0L,0L);
        fx = (float) 0.;
        pt.x = 0L;
        for (i=0;i<256;i++) {
            pt.y = 0L;
            GpiMove(hps,&pt);
            fx += incx;
            pt.x = (long) fx;
            pt.y += (long) ( (float) height * inte[i]);
            GpiBox(hps,DRO_OUTLINE,&pt,0L,0L);
            }
        WinEndPaint(hps);
        return 0;
    case WM_ERASEBACKGROUND:
        return 1;
    }
return WinDefWindowProc(hwnd,msg,mp1,mp2);
}
```

```
/* ----- File IMAGE.C ----- */

#define INCL_WIN
#include <os2.h>
#include <stdio.h>
#include <string.h>
#include "dialog.h"
#include "menu.h"
#include "procdef.h"
#include "util.h"

void far _loadds PMerr(char *);
void enlarge_main(void);
void zoom_main(void);
void enhred_main(void);
void equalize_main(void);
void roberts_main(void);
void sobel_main(void);
void prewitt_main(void);
void kirsch_main(void);
void init_thresh(unsigned char);
void call_thresh(unsigned char);
void end_thresh(int);
void average(void);
void inverse(void);
void texaverage(void);
void calc_hist(float *,unsigned int,unsigned int);
void integ(USHORT,USHORT,USHORT);
void fft(void);
void bfft(int);


extern float histi[256],hists[256],histh[256];
extern float integ_h[256],integ_v[256];
unsigned int act_buf=0x07; /* bits indicate active buffers */

#define BUSY WinSetPointer(HWND_DESKTOP,WinQuerySysPointer(HWND_DESKTOP,SPTR_WAIT,FALSE))
#define NOTBUSY WinSetPointer(HWND_DESKTOP,WinQuerySysPointer(HWND_DESKTOP,SPTR_ARROW,FALSE));
#define BNOTB(x) BUSY; x; NOTBUSY
#define max(a,b)  ( (a) < (b) ) ? (b) : (a)
#define min(a,b)  ( (a) < (b) ) ? (a) : (b)

SHORT           ParseFileName (CHAR *, CHAR *) ;

char szFileName [80] ;

int ha;
int thresh=128;
char buf[5];

MRESULT EXPENTRY ClientWndProc(HWND,USHORT,MPARAM,MPARAM);
MRESULT EXPENTRY HistoWndProc(HWND,USHORT,MPARAM,MPARAM);
MRESULT EXPENTRY IntWndProc(HWND,USHORT,MPARAM,MPARAM);
MRESULT EXPENTRY AboutDlgProc(HWND,USHORT,MPARAM,MPARAM);
MRESULT EXPENTRY OpenDlgProc(HWND,USHORT,MPARAM,MPARAM);
MRESULT EXPENTRY ThreshDlgProc(HWND,USHORT,MPARAM,MPARAM);
MRESULT EXPENTRY BufDlgProc(HWND,USHORT,MPARAM,MPARAM);
MRESULT EXPENTRY IntDlgProc(HWND,USHORT,MPARAM,MPARAM);
MRESULT EXPENTRY HstDlgProc(HWND,USHORT,MPARAM,MPARAM);
HWND hwndFrame,hwndClient;
HAB hab;
HMQ hmq;
QMSG qmsg;
static CHAR szClientClass[]="Image";
static CHAR szHistoClass[]="HistoClass";
static CHAR szIntClass[]="IntClass";
static CHAR title[]="IMAGine -- Inspection Software";
static USHORT temp;

int main(void) {
    hab=WinInitialize(0);
    hmq=WinCreateMsgQueue(hab,0);

    WinRegisterClass(hab,szIntClass,IntWndProc,0L,
        sizeof(float *)+sizeof(USHORT));
    WinRegisterClass(hab,szClientClass,ClientWndProc,0L,0);
    WinRegisterClass(hab,szHistoClass,HistoWndProc,0L,
        sizeof(float *)+3*sizeof(USHORT));

    hwndFrame = WinLoadDlg(HWND_DESKTOP,HWND_DESKTOP,NULL,(HMODULE)NULL,ID_FRAME,NULL);

    WinLoadMenu(hwndFrame,(HMODULE)NULL,ID_RESOURCE);

    WinPostMsg(hwndFrame,WM_UPDATEFRAME,MPFROM2SHORT(FCF_MENU,0),NULL);

    WinSetWindowText(hwndFrame,title);

    WinSetFocus(HWND_DESKTOP,WinWindowFromID(hwndFrame,FID_CLIENT));

    au_init(&ha);
    au_err_msgs(ha,AU_ERR_QUIT|AU_ERR_PRINT,PMerr);
    au_output(ha,0x8000);

    while (WinGetMsg(hab,&qmsg,NULL,0,0))
        WinDispatchMsg(hab,&qmsg);

    WinDestroyWindow(hwndFrame);
    WinDestroyMsgQueue(hmq);
    WinTerminate(hab);
    au_output(ha,0x0000);
    au_end(ha);
    return 0;
    }

MRESULT EXPENTRY ClientWndProc(HWND hwnd, USHORT msg,MPARAM mp1,MPARAM mp2)
{
switch(msg) {
    case WM_CREATE:
        WinPostMsg(WinWindowFromID(hwnd,IDB_64),BM_SETCHECK,
            MPFROM2SHORT(TRUE,0),NULL);
        return 0;
    case WM_COMMAND:
        switch(COMMANDMSG(&msg)->cmd) {
                case IDM_EXIT:
                    WinPostMsg(hwnd,WM_QUIT,0L,0L);
                    return 0;
            case IDM_HUE:
            case IDM_SAT:
            case IDM_INT:
                 temp = COMMANDMSG(&msg)->cmd - IDM_INT;
                WinShowWindow(WinLoadDlg(HWND_DESKTOP,hwnd,HstDlgProc,(HMODULE)NULL,
                    IDD_HISTO,(PVOID) &temp),TRUE);
                EnableMenuItem(hwndFrame,COMMANDMSG(&msg)->cmd,FALSE);
                return 0;
            case IDM_ACQUIRE:
                    au_output(ha,0x8188);
                    WinMessageBox(HWND_DESKTOP,hwndFrame,
                        "Now in passthru mode.\nPosition Camera and object.\nSelect OK to freeze
image.",
                        "IMAGine",0,MB_OK|MB_ICONASTERISK|MB_MOVEABLE);
                    au_output(ha,0x8000);
                    return 0;
        case IDM_ACQUIRERGB:
                    au_output(ha,0x818C);
                    WinMessageBox(HWND_DESKTOP,hwndFrame,
                        "Now in RGB passthru mode.\nPosition Camera and object.\nSelect OK to freeze
image.",
                        "IMAGine",0,MB_OK|MB_ICONASTERISK|MB_MOVEABLE);
                    au_output(ha,0x8002);
                    return 0;
            case IDM_SAVEAS:
            case IDM_OPEN:
                WinDlgBox(HWND_DESKTOP,hwnd,OpenDlgProc,(HMODULE)NULL,
                        IDD_FILEIO,(PVOID) (COMMANDMSG(&msg)->cmd));
                return 0;
            case IDM_BUFFERS:
                WinDlgBox(HWND_DESKTOP,hwnd,BufDlgProc,(HMODULE)NULL,IDD_BUF,NULL);
                return 0;
            case IDM_ABOUT:
                WinDlgBox(HWND_DESKTOP,hwnd,AboutDlgProc,(HMODULE)NULL,IDD_ABOUT,NULL);
                return 0;
            case IDM_THRESHOLD:
                WinShowWindow(WinLoadDlg(HWND_DESKTOP,hwnd,ThreshDlgProc,
                    (HMODULE)NULL,IDD_SCROLL,NULL),TRUE);
                EnableMenuItem(hwndFrame,COMMANDMSG(&msg)->cmd,FALSE);
                return 0;
            case IDM_ENLARGE:
                BNOTB(enlarge_main());
                return 0;
            case IDM_ZOOM:
                BNOTB(zoom_main());
                return 0;
            case IDM_ENHRED:
                BNOTB(enhred_main());
                return 0;
            case IDM_EQUALIZE:
                BNOTB(equalize_main());
                return 0;
            case IDM_AVERAGE:
                BNOTB(average());
                 return 0;
            case IDM_INVERSE:
                BNOTB(inverse());
                return 0;
        case IDM_HOR:
        case IDM_VER:
                temp = COMMANDMSG(&msg)->cmd - IDM_HOR;
                WinShowWindow(WinLoadDlg(HWND_DESKTOP,hwnd,IntDlgProc,(HMODULE)NULL,
                    IDD_INTEGRATE,(PVOID) &temp),TRUE);
                EnableMenuItem(hwndFrame,COMMANDMSG(&msg)->cmd,FALSE);
                return 0;
            case IDM_FOURIER:
                BNOTB(fft());
                return 0;
            case IDM_BAND_8:
                BNOTB(bfft(8));
                return 0;
            case IDM_BAND_16:
                BNOTB(bfft(16));
                return 0;
            case IDM_ROBERTS:
                BNOTB(roberts_main());
                return 0;
            case IDM_SOBEL:
                BNOTB(sobel_main());
                return 0;
            case IDM_PREWITT:
                BNOTB(prewitt_main());
                return 0;
            case IDM_KIRSCH:
                BNOTB(kirsch_main());
                return 0;
            }
            break;
        case WM_ERASEBACKGROUND:
          return (MRESULT)1;
        }
    return WinDefWindowProc(hwnd,msg,mp1,mp2);
    }

MRESULT EXPENTRY AboutDlgProc(HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2)
{
switch (msg) {
    case WM_COMMAND:
        switch(COMMANDMSG(&msg)->cmd) {
            case DID_OK:
            case DID_CANCEL:
                WinDismissDlg(hwnd,TRUE);
                return 0;
            }
        break;
    }
return WinDefDlgProc(hwnd,msg,mp1,mp2);
}

MRESULT EXPENTRY IntDlgProc(HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2)
{
USHORT t;
FILE *fp;
float *intar;
int cpos,numl;
static CHAR *title[2]={"Horizontal Integration","Vertical Integration"};
static float *ptr[2]={integ_h,integ_v};
HWND win_hwnd = WinWindowFromID(hwnd,IDW_BAR);
switch (msg) {
    case WM_INITDLG:
        t = *(USHORT *) PVOIDFROMMP(mp2);
            WinSetWindowText(WinWindowFromID(hwnd,IDT_INT),"0");
            WinSetWindowText(WinWindowFromID(hwnd,IDE_INT),"512");
        WinSetWindowText(WinWindowFromID(hwnd,FID_TITLEBAR),title[t]);
        WinSetWindowPtr(win_hwnd,0,(PVOID)ptr[t]);
        WinSetWindowUShort(win_hwnd,4,t);
        WinSendDlgItemMsg(hwnd,IDS_INT,SBM_SETSCROLLBAR,
            MPFROMSHORT(0),MPFROM2SHORT(0,511));
        WinSendDlgItemMsg(hwnd,IDS_INT,SBM_SETPOS,
            MPFROM2SHORT(0,0),NULL);
        return 0;
    case WM_VSCROLL:
            cpos = WinSendDlgItemMsg(hwnd,IDS_INT,SBM_QUERYPOS,0L,0L);
        switch (SHORT2FROMMP(mp2)) {
            case SB_LINEDOWN:
                cpos = min(511,cpos+1);
                break;
            case SB_LINEUP:
                cpos = max(0,cpos-1);
                break;
            case SB_PAGEDOWN:
                cpos = min(511,cpos+16);
                break;
            case SB_PAGEUP:
                cpos = max(0,cpos-16);
                break;
            case SB_SLIDERTRACK:
                cpos = SHORT1FROMMP(mp2);
                break;
            default:
                return 0;
            }
            WinQueryDlgItemShort(hwnd,IDE_INT,&numl,FALSE);
            if (cpos+numl > 512) cpos=512-numl;
        WinSendDlgItemMsg(hwnd,IDS_INT,SBM_SETPOS,MPFROM2SHORT(cpos,0),NULL);
        sprintf(buf,"%d",cpos);
        WinSetDlgItemText(hwnd,IDT_INT,buf);
            if (WinSendDlgItemMsg(hwnd,IDB_AUTO,BM_QUERYCHECK,0L,0L)==1)
                WinPostMsg(hwnd,WM_COMMAND,MPFROMSHORT(DID_OK),mp2);
        return 0;
    case WM_COMMAND:
        switch(COMMANDMSG(&msg)->cmd) {
            case IDB_CANCEL:
                EnableMenuItem(hwndFrame,IDM_HOR+
                    WinQueryWindowUShort(win_hwnd,4),TRUE);
                WinDismissDlg(hwnd,TRUE);
                return 0;
            case IDB_OK:
                        cpos = WinSendDlgItemMsg(hwnd,IDS_INT,SBM_QUERYPOS,0L,0L);
                        WinQueryDlgItemShort(hwnd,IDE_INT,&numl,FALSE);
                        if (cpos+numl > 512) numl = 512-cpos;
                BNOTB(integ(WinQueryWindowUShort(win_hwnd,4),cpos,numl));
                WinInvalidateRect(win_hwnd,NULL,FALSE);
                return 0;
                case IDB_SAVE:
                if (WinDlgBox(HWND_DESKTOP,hwnd,OpenDlgProc,(HMODULE)NULL,
                            IDD_FILEIO,(PVOID)COMMANDMSG(&msg)->cmd)) { /* save file now */
                            if ((fp = fopen(szFileName,"w")) == NULL) {
                                WinMessageBox(HWND_DESKTOP,hwndFrame,"Could not open file!",
                                    "IMAGine",0,MB_OK|MB_ICONEXCLAMATION|MB_MOVEABLE);
                                return 0;
                                }
                            intar = (float*) WinQueryWindowPtr(win_hwnd,0);
                            for (t=0;t<256;t++)
                                fprintf(fp,"%.6f\n",intar[t]);
                            fclose(fp);
                            }
                        return 0;
            }
        break;
    }
return WinDefDlgProc(hwnd,msg,mp1,mp2);
}

MRESULT EXPENTRY HstDlgProc(HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2)
{
static CHAR *title[3]={"Intensity Histogram","Saturation Histogram",
    "Hue Histogram"};
static float *ptr[3]={histi,hists,histh};
USHORT t;
HWND win_hwnd;
switch (msg) {
    case WM_INITDLG:
            win_hwnd = WinWindowFromID(hwnd,IDW_BAR);
        t = *(USHORT *) PVOIDFROMMP(mp2);
        WinSetWindowText(WinWindowFromID(hwnd,FID_TITLEBAR),title[t]);
        WinSetWindowPtr(win_hwnd,0,(PVOID)ptr[t]);
        WinSetWindowUShort(win_hwnd,4,t);
        WinSetWindowUShort(win_hwnd,6,8);
        WinSetWindowUShort(win_hwnd,8,256);
        return 0;
    case WM_CONTROL:
            win_hwnd = WinWindowFromID(hwnd,IDW_BAR);
        WinSetWindowUShort(win_hwnd,6,SHORT1FROMMP(mp1) -IDB_16 + 4);
        return 0;
    case WM_COMMAND:
        switch(COMMANDMSG(&msg)->cmd) {
            case DID_CANCEL:
                        win_hwnd = WinWindowFromID(hwnd,IDW_BAR);
                        EnableMenuItem(hwndFrame,IDM_INT+
                WinQueryWindowUShort(win_hwnd,4),TRUE);
                WinDismissDlg(hwnd,TRUE);
                return 0;
            case DID_OK:
                        win_hwnd = WinWindowFromID(hwnd,IDW_BAR);
                        t = WinQueryWindowUShort(win_hwnd,6);
                WinSetWindowUShort(win_hwnd,8, 1 << t);
                BNOTB(calc_hist((float *)WinQueryWindowPtr(win_hwnd,0),
                WinQueryWindowUShort(win_hwnd,4),t));
                WinInvalidateRect(win_hwnd,NULL,FALSE);
                return 0;
            }
        break;
    }
return WinDefDlgProc(hwnd,msg,mp1,mp2);
}

MRESULT EXPENTRY BufDlgProc(HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2)
{
int i;
switch (msg) {
    case WM_INITDLG:
        for (i=0;i<4;i++)
            if (act_buf & (1<<i))
                WinSendDlgItemMsg(hwnd,IDC_INT+i,BM_SETCHECK,MPFROMSHORT(1),0L);
        return 0;
    case WM_COMMAND:
        switch(COMMANDMSG(&msg)->cmd) {
            case DID_OK:
                act_buf = 0;
                for (i=0;i<4;i++)
                    act_buf |= (WinSendDlgItemMsg(hwnd,IDC_INT+i,BM_QUERYCHECK,
                        NULL,NULL) ? (1<<i) : 0);
                if (WinSendDlgItemMsg(hwnd,IDC_CLR,BM_QUERYCHECK,NULL,NULL))
                    for (i=0;i<4;i++)
                        if (!(act_buf & (1<<i))) au_buf_clear(ha,i);
                WinDismissDlg(hwnd,TRUE);
                return 0;
            }
        break;
    }
```

```
return WinDefDlgProc(hwnd,msg,mp1,mp2);
}

MRESULT EXPENTRY ThreshDlgProc(HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2)
{
switch (msg) {
    case WM_INITDLG:
        WinSendDlgItemMsg(hwnd,IDD_SCROLLBAR,SBM_SETSCROLLBAR,
            MPFROMSHORT(thresh),MPFROM2SHORT(0,255));
        WinSendDlgItemMsg(hwnd,IDD_SCROLLBAR,SBM_SETPOS,
            MPFROM2SHORT(thresh,0),NULL);
        sprintf(buf,"%d",thresh);
        WinSetDlgItemText(hwnd,IDD_SCROLLNUM,buf);
        BNOTB(init_thresh( (unsigned char) thresh));
        return 0;
    case WM_VSCROLL:
        switch (SHORT2FROMMP(mp2)) {
            case SB_LINEDOWN:
                thresh = min(255,thresh+1);
                break;
            case SB_LINEUP:
                thresh = max(0,thresh-1);
                break;
            case SB_PAGEDOWN:
                thresh = min(255,thresh+16);
                break;
            case SB_PAGEUP:
                thresh = max(0,thresh-16);
                break;
            case SB_SLIDERTRACK:
                thresh = SHORT1FROMMP(mp2);
                break;
            default:
                return 0;
            }
        WinSendDlgItemMsg(hwnd,IDD_SCROLLBAR,SBM_SETPOS,
            MPFROM2SHORT(thresh,0),NULL);
        sprintf(buf,"%d",thresh);
        WinSetDlgItemText(hwnd,IDD_SCROLLNUM,buf);
        return 0;
    case WM_COMMAND:
        switch(COMMANDMSG(&msg)->cmd) {
            case DID_OK:
                BNOTB(end_thresh(1));
                WinDismissDlg(hwnd,TRUE);
                EnableMenuItem(hwndFrame,IDM_THRESHOLD,TRUE);
                return 0;
            case DID_CANCEL:
                BNOTB(end_thresh(0));
                WinDismissDlg(hwnd,TRUE);
                EnableMenuItem(hwndFrame,IDM_THRESHOLD,TRUE);
                return 0;
            case IDB_UPDATE:
                BNOTB(call_thresh( (unsigned char) thresh));
                return 0;
            }
        break;
    }
return WinDefDlgProc(hwnd,msg,mp1,mp2);
}


void far _loadds PMerr(str)
char *str;
{
WinAlarm(HWND_DESKTOP,WA_NOTE);
WinMessageBox(HWND_DESKTOP,hwndFrame,str,NULL,0,MB_OK | MB_ICONEXCLAMATION);
}


VOID FillDirListBox (HWND hwnd, CHAR *pcCurrentPath)
     {
     static CHAR szDrive [] = "  :" ;
     FILEFINDBUF findbuf ;
     HDIR        hDir = 1 ;
     SHORT       sDrive ;
     USHORT      usDriveNum, usCurPathLen, usSearchCount = 1 ;
     ULONG       ulDriveMap ;

     DosQCurDisk (&usDriveNum, &ulDriveMap) ;
     pcCurrentPath [0] = (CHAR) usDriveNum + (CHAR)'@' ;
     pcCurrentPath [1] = ':' ;
     pcCurrentPath [2] = '\\' ;
     usCurPathLen = 64 ;
     DosQCurDir (0, pcCurrentPath + 3, &usCurPathLen) ;

     WinSetDlgItemText (hwnd, IDD_PATH, pcCurrentPath) ;
     WinSendDlgItemMsg (hwnd, IDD_DIRLIST, LM_DELETEALL, NULL, NULL) ;

     for (sDrive = 0 ; sDrive < 26 ; sDrive++)
          if (ulDriveMap & 1L << sDrive)
               {
               szDrive [1] = (CHAR) sDrive + (CHAR)'A' ;

               WinSendDlgItemMsg (hwnd, IDD_DIRLIST, LM_INSERTITEM,
                                   MPFROM2SHORT (LIT_END, 0),
                                   MPFROMP (szDrive)) ;
               }

     DosFindFirst ("*.*", &hDir, 0x0017, &findbuf, sizeof findbuf,
                              &usSearchCount, 0L) ;
     while (usSearchCount)
          {
          if (findbuf.attrFile & 0x0010 &&
                    (findbuf.achName [0] != '.' || findbuf.achName [1]))

               WinSendDlgItemMsg (hwnd, IDD_DIRLIST, LM_INSERTITEM,
                                   MPFROM2SHORT (LIT_SORTASCENDING, 0),
                                  MPFROMP (findbuf.achName)) ;

          DosFindNext (hDir, &findbuf, sizeof findbuf, &usSearchCount) ;
          }
     }

VOID FillFileListBox (HWND hwnd)
     {
     FILEFINDBUF findbuf ;
     HDIR        hDir = 1 ;
     USHORT      usSearchCount = 1 ;

     WinSendDlgItemMsg (hwnd, IDD_FILELIST, LM_DELETEALL, NULL, NULL) ;

     DosFindFirst ("*.*", &hDir, 0x0007, &findbuf, sizeof findbuf,
                             &usSearchCount, 0L) ;
     while (usSearchCount)
          {
          WinSendDlgItemMsg (hwnd, IDD_FILELIST, LM_INSERTITEM,
                            MPFROM2SHORT (LIT_SORTASCENDING, 0),
                            MPFROMP (findbuf.achName)) ;

          DosFindNext (hDir, &findbuf, sizeof findbuf, &usSearchCount) ;
          }
     }

MRESULT EXPENTRY OpenDlgProc (HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2)
     {
     static CHAR szCurrentPath [80], szBuffer [80] ;
     SHORT       sSelect ;
     static (far pascal *sfunc)(int,char *);
     switch (msg)
          {
          case WM_INITDLG:
               FillDirListBox (hwnd, szCurrentPath) ;
               FillFileListBox (hwnd) ;
               switch( (int) mp2) {
                    case IDM_OPEN:
                        sfunc = au_load;
                        WinSetDlgItemText(hwnd,IDD_HEAD,"Open File");
                        WinSetDlgItemText(hwnd,DID_OK,"Open");
                        break;
                    case IDM_SAVEAS:
                        sfunc = au_save;
                        WinSetDlgItemText(hwnd,IDD_HEAD,"Save File");
                        WinSetDlgItemText(hwnd,DID_OK,"Save");
                        break;
                    case IDB_SAVE:
                 sfunc = NULL;
                        WinSetDlgItemText(hwnd,IDD_HEAD,"Save to Neural Net File");
                        WinSetDlgItemText(hwnd,DID_OK,"Save");
                        break;
                    }
               WinSendDlgItemMsg (hwnd, IDD_FILEEDIT, EM_SETTEXTLIMIT,
                               MPFROM2SHORT (80, 0), NULL) ;
     return 0 ;

case WM_CONTROL:
     if (SHORT1FROMMP (mp1) == IDD_DIRLIST ||
         SHORT1FROMMP (mp1) == IDD_FILELIST)
          {
          sSelect = (USHORT) WinSendDlgItemMsg (hwnd,
                                         SHORT1FROMMP (mp1),
                                         LM_QUERYSELECTION, 0L, 0L) ;

          WinSendDlgItemMsg (hwnd, SHORT1FROMMP (mp1),
                             LM_QUERYITEMTEXT,
                             MPFROM2SHORT (sSelect, sizeof szBuffer),
                             MPFROMP (szBuffer)) ;
          }

     switch (SHORT1FROMMP (mp1))             // Control ID
          {
          case IDD_DIRLIST:
               switch (SHORT2FROMMP (mp1))   // notification code
                    {
                    case LN_ENTER:
                         if (szBuffer [0] == ' ')
                              DosSelectDisk (szBuffer [1] - '@') ;
                         else
                              DosChDir (szBuffer, 0L) ;

                         FillDirListBox (hwnd, szCurrentPath) ;
                         FillFileListBox (hwnd) ;

                         WinSetDlgItemText (hwnd, IDD_FILEEDIT, "") ;
                         return 0 ;
                    }
               break ;

          case IDD_FILELIST:
               switch (SHORT2FROMMP (mp1))   // notification code
                    {
                    case LN_SELECT:
                         WinSetDlgItemText (hwnd, IDD_FILEEDIT,
                                             szBuffer) ;
                         return 0 ;

                    case LN_ENTER:
                         ParseFileName (szFileName, szBuffer) ;
                         WinDismissDlg (hwnd, TRUE) ;
                                if (sfunc != NULL) {
                                        BNOTB((*sfunc)(ha,szFileName));
                                      }
                                return 0 ;
                    }
               break ;
          }
```

```
                break ;

           case WM_COMMAND:
                switch (COMMANDMSG(&msg)->cmd)
                     {
                     case DID_OK:
                          WinQueryDlgItemText (hwnd, IDD_FILEEDIT,
                                               sizeof szBuffer, szBuffer) ;
                          switch (ParseFileName (szCurrentPath, szBuffer))
                               {
                               case 0:
                                    WinAlarm (HWND_DESKTOP, WA_ERROR) ;
                                    FillDirListBox (hwnd, szCurrentPath) ;
                                    FillFileListBox (hwnd) ;
                                    return 0 ;

                               case 1:
                                    FillDirListBox (hwnd, szCurrentPath) ;
                                    FillFileListBox (hwnd) ;
                                    WinSetDlgItemText (hwnd, IDD_FILEEDIT, "") ;
                                    return 0 ;

                               case 2:
                                    strcpy (szFileName, szCurrentPath) ;
                                    WinDismissDlg (hwnd, TRUE) ;
                                    if (sfunc != NULL) {
                                        BNOTB((*sfunc)(ha,szFileName));
                                        }
                                    return 0 ;
                               }
                          break ;

                     case DID_CANCEL:
                          WinDismissDlg (hwnd, FALSE) ;
                          return 0 ;
                     }
                break ;
           }
      return WinDefDlgProc (hwnd, msg, mp1, mp2) ;
      }

SHORT ParseFileName (CHAR *pcOut, CHAR *pcIn)
      {
           /*-------------------------------------------------------------
              Input:    pcOut -- Pointer to parsed file specification.
                        pcIn  -- Pointer to raw file specification.

              Returns:  0 -- pcIn had invalid drive or directory.
                        1 -- pcIn was empty or had no filename.
                        2 -- pcOut points to drive, full dir, and file name.

              Changes current drive and directory per pcIn string.
             -------------------------------------------------------------*/
```

```
CHAR   *pcLastSlash, *pcFileOnly ;
ULONG  ulDriveMap ;
USHORT usDriveNum, usDirLen = 64 ;

strupr (pcIn) ;

          // If input string is empty, return 1

if (pcIn [0] == '\0')
     return 1 ;

          // Get drive from input string or current drive

if (pcIn [1] == ':')
     {
     if (DosSelectDisk (pcIn [0] - '@'))
          return 0 ;

     pcIn += 2 ;
     }
DosQCurDisk (&usDriveNum, &ulDriveMap) ;

*pcOut++ = (CHAR) usDriveNum + (CHAR)'@' ;
*pcOut++ = ':' ;
*pcOut++ = '\\' ;

          // If rest of string is empty, return 1

if (pcIn [0] == '\0')
     return 1 ;

          // Search for last backslash.  If none, could be directory.

if (NULL == (pcLastSlash = strrchr (pcIn, '\\')))
     {
     if (!DosChDir (pcIn, 0L))
          return 1 ;

               // Otherwise, get current dir & attach input filename

     DosQCurDir (0, pcOut, &usDirLen) ;

     if (strlen (pcIn) > 12)
          return 0 ;

     if (*(pcOut + strlen (pcOut) - 1) != '\\')
          strcat (pcOut++, "\\") ;

     strcat (pcOut, pcIn) ;
     return 2 ;
     }
          // If the only backslash is at beginning, change to root

if (pcIn == pcLastSlash)
```

```
          {
          DosChDir ("\\", 0L) ;

          if (pcIn [1] == '\0')
               return 1 ;

          strcpy (pcOut, pcIn + 1) ;
          return 2 ;
          }
               // Attempt to change directory -- Get current dir if OK

     *pcLastSlash = '\0' ;

     if (DosChDir (pcIn, 0L))
          return 0 ;

     DosQCurDir (0, pcOut, &usDirLen) ;

               // Append input filename, if any

     pcFileOnly = pcLastSlash + 1 ;

     if (*pcFileOnly == '\0')
          return 1 ;

     if (strlen (pcFileOnly) > 12)
          return 0 ;

     if (*(pcOut + strlen (pcOut) - 1) != '\\')
          strcat (pcOut++, "\\") ;

     strcat (pcOut, pcFileOnly) ;
     return 2 ;
     }

void EnableMenuItem(HWND hwndParent,SHORT sMenuItem, BOOL fEnable)
{
HWND hwndMenu = WinWindowFromID(hwndParent,FID_MENU);

WinSendMsg(hwndMenu,MM_SETITEMATTR,MPFROM2SHORT(sMenuItem,TRUE),
    MPFROM2SHORT(MIA_DISABLED, fEnable ? 0 : MIA_DISABLED));
}
```

```
/* ----- File INTEG.C ----- */

#include <stdio.h>
#include "util.h"

#define HEIGHT 512
#define WIDTH 512
#define max2(a,b) ((a) > (b)) ? (a) : (b)
extern unsigned char huge ipic[HEIGHT][WIDTH];
extern unsigned char huge opic[HEIGHT][WIDTH];
float integ_h[256],integ_v[256];
void integv(unsigned int,unsigned int);
void integh(unsigned int,unsigned int);
void integ(unsigned int,unsigned int,unsigned int);

extern int ha;

void integ(or,start,num)
unsigned int or,start,num;
{
switch (or) {
    case 0:
        integh(start,num);
        break;
    case 1:
        integv(start,num);
        break;
    }
}

void integh(unsigned int start,unsigned int num)
{
unsigned int i,j;
float mx= (float) 0.;
au_readscr(ha,ipic,0);
for (i=0;i<256;i++) integ_h[i]= (float) 0.;
for (i=0;i<512;i++)
    for (j=start;j<start+num;j++) integ_h[i/2]+= (float) ipic[i][j];
for (i=0;i<256;i++) mx = max2(mx,integ_h[i]);
if (mx != (float) 0.)
    for (i=0;i<256;i++) integ_h[i] /= mx;
}

void integv(unsigned int start,unsigned int num)
{
unsigned int i,j;
float mx= (float) 0.;
au_readscr(ha,ipic,0);
for (i=0;i<256;i++) integ_v[i]= (float) 0.;
for (i=start;i<start+num;i++)
    for (j=0;j<512;j++)
      integ_v[j/2]+= (float) ipic[i][j];
for (i=0;i<256;i++) mx = max2(mx,integ_v[i]);
if (mx != (float)0.)
    for (i=0;i<256;i++) integ_v[i] /= mx;
}
```

```
/* ----- File MAPC.C ----- */

;       Static Name Aliases
;
        TITLE   mapn.c
        NAME    mapn

        .286p
        .287
MAPN_TEXT        SEGMENT  WORD PUBLIC 'CODE'
MAPN_TEXT        ENDS
_DATA   SEGMENT  WORD PUBLIC 'DATA'
_DATA   ENDS
CONST   SEGMENT  WORD PUBLIC 'CONST'
CONST   ENDS
_BSS    SEGMENT  WORD PUBLIC 'BSS'
_BSS    ENDS
DGROUP  GROUP   CONST, _BSS, _DATA
        ASSUME  CS: MAPN_TEXT, DS: DGROUP, SS: DGROUP
EXTRN   __acrtused:ABS
EXTRN   __AHINCR:FAR
MAPN_TEXT      SEGMENT
        ASSUME  CS: MAPN_TEXT
; Line 4
        PUBLIC  _mapn
_mapn   PROC FAR
        enter   WORD PTR 22,0
        push    di
        push    si
; Line 6
;       pic = 6
;       map = 10
;       li = -8
;       i = -10
;       tpic = -4
; Line 10
        mov     WORD PTR [bp-10],0       ;i
        mov     WORD PTR [bp-8],0        ;li
        mov     WORD PTR [bp-6],1
        mov     ax,WORD PTR [bp+6]       ;pic
        mov     dx,WORD PTR [bp+8]
        mov     WORD PTR [bp-20],ax
        mov     WORD PTR [bp-18],dx
        mov     di,4
        add     WORD PTR [bp-10],4       ;i
$L20002:
; Line 11
        mov     ax,WORD PTR [bp-20]
        mov     dx,WORD PTR [bp-18]
        mov     WORD PTR [bp-4],ax       ;tpic
        mov     WORD PTR [bp-2],dx
; Line 12
        mov     WORD PTR [bp-14],0
        mov     WORD PTR [bp-12],1
        mov     WORD PTR [bp-16],ds

        les     si,DWORD PTR [bp-4]      ;tpic
;        lds     si,DWORD PTR [bp-4]      ;tpic
        lds     bx,DWORD PTR [bp+10]     ;map
        mov     cx,65535

$L20000:
; Line 13
        mov     al,BYTE PTR es:[si]      ;al = *tpic
        xlatb                            ;al = map[al]
        mov     BYTE PTR es:[si],al      ;*tpic = al
        inc     si
        loop    $L20000
        mov     al,BYTE PTR es:[si]
        xlatb
        mov     BYTE PTR es:[si],al

        mov     WORD PTR [bp-4],si       ;tpic
        mov     WORD PTR [bp-2],es
        mov     ds,WORD PTR [bp-16]
        ASSUME DS: NOTHING
; Line 14
        mov     ax,OFFSET __AHINCR
        add     WORD PTR [bp-18],ax
        dec     di
        jne     $L20002
; Line 15
        pop     si
        pop     di
        leave
        ret

_mapn   ENDP
MAPN_TEXT        ENDS
END
```

```
/* ----- File PROCESS.C ----- */

#include "util.h"
#define LORED 255
#define HIRED 240
#define SAT_THRESH 32

unsigned char huge tpic[512][512];
unsigned char huge ipic[512][512];
unsigned char huge opic[512][512];
static unsigned int ihist[256];
static unsigned char map[256];
void zoom_main(void);
void enlarge_main(void);
void enhred_main(void);
void equalize_main(void);
void zoom(int);
void enlarge(int buf);
void init_thresh(unsigned char th);
void call_thresh(unsigned char th);
void end_thresh(int);
void average(void);
void mapn(unsigned char huge *,unsigned char *);
void copy(unsigned char huge *,unsigned char huge *);

extern int ha;
extern unsigned int act_buf;

void zoom_main()
{
int i;
        /* Initialization */
for (i=0;i<4;i++)
    if (act_buf & (1<<i))
        zoom(i);
}

void zoom(buff)
int buff;
{
int i,j,x,y;
au_readscr(ha,ipic,buff);
for (i=0 , x=128 ;i<512;i+=2 , x++)
        for (j=0 , y=128;j<512;j+=2 , y++) {
                opic[i][j] = ipic[x][y];
                opic[i][j+1]= (ipic[x][y]+ipic[x][y+1])/2;
                opic[i+1][j]= (ipic[x][y]+ipic[x+1][y])/2;
                opic[i+1][j+1]= (ipic[x][y]+ipic[x+1][y+1])/2;
                }
au_writescr(ha,opic,buff);
}

void enlarge_main()
```

```
{
int i;
        /* Initialization */

for (i=0;i<4;i++)
    if (act_buf & (1<<i))
        enlarge(i);
}

void enlarge(buf)
int buf;
{
int i,j;
unsigned char huge *topic1 = (unsigned char huge *) opic[0];
unsigned char huge *topic2 = (unsigned char huge *) opic[1];
unsigned char huge *tipic = (unsigned char huge *) &ipic[128][128];
au_readscr(ha,ipic,buf);
for (i=0;i<256;i++,topic1=topic2,topic2+=512,tipic+=256)
    for (j=0;j<256;j++,topic1+=2,topic2+=2)
        *(topic1) = *(topic1+1) = *(topic2) = *(topic2+1) = *(tipic++);
au_writescr(ha,opic,buf);
}

void enhred_main()
{
long li;
unsigned char huge *topic = (unsigned char huge *) opic;
unsigned char huge *tipic = (unsigned char huge *) ipic;
unsigned char huge *ttpic = (unsigned char huge *) tpic;
au_buf_clear(ha,3);
au_readscr(ha,tpic,0);
au_readscr(ha,ipic,2);
au_readscr(ha,opic,1);
for (li=0L;li<262144L;li++,tipic++,topic++,ttpic++)
    if (((*tipic <= LORED) || (*tipic >= HIRED)) && (*topic > SAT_THRESH)) {
        *ttpic = 255;
        *tipic = 255;
        *topic = 255;
        }
    else {
        *ttpic = 0;
        }
ttpic = (unsigned char huge *) &tpic[480][0];
for (li=0L;li<32L*512L;li++,ttpic++)
    *ttpic = 0;
au_writescr(ha,tpic,0);
au_writescr(ha,ipic,2);
au_writescr(ha,opic,1);
}

void equalize_main()
{
const unsigned int bins=256;
unsigned int binpos=0;
unsigned int i;
unsigned long sum=0L;
unsigned long maxbinval=1024L;
unsigned long li;
const unsigned long binwidth=1024L;
unsigned char huge *tp = ipic[0];
/*
maxbinval=binwidth=(512L*512L)/(unsigned long) bins;
*/
au_readscr(ha,ipic,0);
for (i=0;i<256;i++) ihist[i]=0;
for (li=0L;li<262144L;li++,tp++)
    ++ihist[(unsigned int)(*tp) ];
for (i=0;i<256;i++) {
    while ((sum+(unsigned long)ihist[i]) > maxbinval) {
        binpos++;
        maxbinval += binwidth;
        }
    map[i]= (unsigned char) (binpos*256/bins);
    sum += (unsigned long) ihist[i];
    }
tp = ipic[0];
mapn(tp,map);
au_writescr(ha,ipic,0);
}


void init_thresh(th)
unsigned char th;
{
unsigned int i;
unsigned char map[256];
unsigned char huge *tipic = (unsigned char huge *) tpic;
unsigned char huge *topic = (unsigned char huge *) opic;
au_readscr(ha,tpic,0);
copy(tipic,topic);
for (i=0;i<256;i++) map[i] = (unsigned char) ((i<th) ? 0:255);
mapn(topic,map);
au_writescr(ha,opic,0);
}

void call_thresh(th)
unsigned char th;
{
unsigned int i;
unsigned char map[256];
unsigned char huge *tipic = (unsigned char huge *) tpic;
unsigned char huge *topic = (unsigned char huge *) opic;
copy(tipic,topic);
for (i=0;i<256;i++) map[i] = (unsigned char) ((i<th) ? 0:255);
mapn(topic,map);
au_writescr(ha,opic,0);
}
```

```
void end_thresh(flag)
int flag;
{
if (!flag)
    au_writescr(ha,tpic,0);
}

void average(void)
{
int i,j,t;
au_readscr(ha,ipic,0);
for (i=1;i<511;i++)
    for (j=1;j<511;j++) {
        t = (int)ipic[i-1][j-1]+(int)ipic[i-1][j]+(int)ipic[i][j-1];
        t += (int)ipic[i][j]+(int)ipic[i][j+1]+(int)ipic[i+1][j]+(int)ipic[i+1][j+1];
        opic[i][j] = (unsigned char) (t/7);
        }
au_writescr(ha,opic,0);
}

void inverse(void)
{
int i,j;
au_readscr(ha,ipic,0);
for (i=1;i<511;i++)
    for (j=1;j<511;j++) {
        opic[i][j] = (unsigned char) (255-ipic[i][j]);
        }
au_writescr(ha,opic,0);
}
```

```
/* ----- File TEXTURE.C ----- */

#include "util.h"
#include <math.h>

#define max2(a,b) ((a) >(b)) ? (a) : (b);
#define Height 512
#define Width 512

extern unsigned char huge ipic[Height][Width];
unsigned char huge Ak[Height][Width];
unsigned char huge Ik[Height][Width];
float hist[256],histi[256],histh[256],hists[256];
extern int ha;
void one_iter(void);
void threshold_regions(void);
void average_pic(void);
unsigned char ret_region_aver(unsigned int,unsigned int,unsigned int);
unsigned int pick_region(unsigned int,unsigned int);
unsigned int pick_spec_region(unsigned int,unsigned int);
unsigned char aver(unsigned int,unsigned int);
unsigned int homogen(unsigned int,unsigned int);
void calc_hist(float *,unsigned int,unsigned int);
void main(void);
void doitall(void);

#ifdef DSGSDFGDFG
void doitall(void)
{
one_iter();
one_iter();
threshold_regions();
}

void one_iter(void)
{
        unsigned int i,j;
        au_readscr(ha,ipic,0);
        average_pic();
        for (i=0; i<Height; i++)
                for(j=0; j<Width; j++)
                        Ik[i][j] = (unsigned char) 0;

        for (i=4; i<Height-4; i++)
                for(j=4; j<Width-4; j++)
                        Ik[i][j] = ret_region_aver(i,j,pick_region(i,j));
        for (i=0; i<Height; i++)
                for(j=0;j<4;j++) {
                        Ik[i][j] = ret_region_aver(i,j,pick_spec_region(i,j));
                        Ik[i][Width-1-j] = ret_region_aver(i,Width-1-j,pick_spec_region(i,Width-1-
j));
                }
        for (i=0; i<4;i++)
```

```
                for (j=4; j<Width-4;j++) {
                        Ik[i][j] = ret_region_aver(i,j,pick_spec_region(i,j));
                        Ik[Height-1-i][j] = ret_region_aver(Height-1-i,j,pick_spec_region(Height-1-
i,j));
                }
        au_writescr(ha,Ik,0);
}
#endif

void calc_hist(hist,buf,hist_level)
float *hist;
unsigned int buf,hist_level;
{
float mx=(float)0.;
unsigned char huge *tp = (unsigned char huge *) ipic;
int i;
unsigned long li;
au_readscr(ha,ipic,buf);
for (i=0;i<256;i++) hist[i] = (float)0.;
for (li=0;li<262144L;li++,tp++)
    hist[ ((unsigned int) (*tp) >> (8-hist_level))] += (float) 1.;
for (i=0;i<256;i++) mx = max2(mx,hist[i]);
for (i=0;i<256;i++) hist[i] /= mx;
}


#ifdef DFGDFGDFGFDGFDG
unsigned char ret_region_aver(i,j,d)
unsigned int i,j,d;
{
switch (d) {
        case 0:
                return(aver(i,j));
                break;
        case 1:
                return(aver(i+2,j+2));
                break;
        case 2:
                return(aver(i-2,j+2));
                break;
        case 3:
                return(aver(i-2,j-2));
                break;
        case 4:
                return(aver(i+2,j-2));
                break;
        }
}

unsigned char aver(x,y)
unsigned int x,y;
{
        unsigned int t;
        t = ipic[x-2][y-2]+ipic[x-1][y-2]+ipic[x][y-2]+ipic[x+1][y-2]+ipic[x+2][y-2];
        t += ipic[x-2][y-1]+ipic[x-1][y-1]+ipic[x][y-1]+ipic[x+1][y-1]+ipic[x+2][y-1];
        t += ipic[x-2][y]+ipic[x-1][y]+ipic[x][y]+ipic[x+1][y]+ipic[x+2][y];
        t += ipic[x-2][y+1]+ipic[x-1][y+1]+ipic[x][y+1]+ipic[x+1][y+1]+ipic[x+2][y+1];
        t += ipic[x-2][y+2]+ipic[x-1][y+2]+ipic[x][y+2]+ipic[x+1][y+2]+ipic[x+2][y+2];
        t /= 25;
return( (unsigned char) t);
}

void threshold_regions(void)
{
        unsigned int i,a,b,j=0,k=0;
        unsigned int valleys[10],peaks[10];
        unsigned int hist_level = 8;
        unsigned int num_hist_levels= 1 << hist_level;
        unsigned int level,low_pt=0,high_pt=0,mode=1;
        unsigned char map[256];
        float low_val= (float) 1., high_val= (float) 0.;
        calc_hist(hist,0,hist_level);
        au_readscr(ha,ipic,0);
        valleys[j++] = 0;
        for (i=0; i<num_hist_levels; i++) {
                switch(mode) {
                        case 0: /* looking for minimum */
                                if (hist[i]<=low_val) {
                                        low_val = hist[i];
                                        low_pt = i;
                                }
                                else if (hist[i] > (float)3.*low_val) {
                                        valleys[j++] = low_pt;
                                        mode = 1;
                                        high_val = hist[i];
                                        high_pt = i;
                                }
                                break;
                        case 1: /* looking for maximum */
                                if (hist[i] >= high_val) {
                                        high_val = hist[i];
                                        high_pt = i;
                                }
                                else if (hist[i]*3. < high_val) {
                                        peaks[k++] = high_pt;
                                        mode = 0;
                                        low_val = hist[i];
                                        low_pt = i;
                                }
                                break;
                        }
                }
        switch(mode) {
                case 0:
                        valleys[j++] = i-1;
                        break;
                case 1:
                        peaks[k++] = high_pt;
                        valleys[j++] = i-1;
                        break;
                }
        b = 0;
        for (i=1;i<j;i++) {
                level = peaks[i-1]*256/num_hist_levels;
                for (a=b; a<(valleys[i]+1)*256/num_hist_levels; a++)
                        map[a] = (unsigned char) level;
                b = a;
        }
        for(i=0;i<Height;i++)
                for(j=0;j<Width;j++) {
                        k = (int) (ipic[i][j]);
                        ipic[i][j] = map[k];
                        }
        au_writescr(ha,ipic,0);
}


unsigned int pick_spec_region(x,y)
unsigned int x,y;
{
unsigned int mnh,th;
unsigned int dir;
mnh = 65535;
if ((x>=2) && (x<=253) && (y>=2) && (y<=253)) {
    if ((th=homogen(x,y))< mnh) {
        mnh = th;
        dir = 0;
        }
    }
if ((x<=251) && (y<=251)) {
    if ((th=homogen(x+2,y+2))< mnh) {
        mnh = th;
        dir = 1;
        }
    }
if ((x>=4) && (y<=251)) {
    if ((th=homogen(x-2,y+2))< mnh) {
        mnh = th;
        dir = 2;
        }
    }
if ((x>=4) && (y>=4)) {
    if ((th=homogen(x-2,y-2))< mnh) {
        mnh = th;
        dir = 3;
        }
    }
if ((x<=251) && (y>=4)) {
    if ((th=homogen(x+2,y-2))< mnh) {
        mnh = th;
        dir = 4;
        }
    }
return(dir);
}

unsigned int pick_region(x,y)
unsigned int x,y;
{
        unsigned int mnh,dir,th;
        mnh = homogen(x,y); dir = 0;
        if ((th=homogen(x+2,y+2))< mnh) {
                mnh = th;
                dir = 1;
        }
        if ((th=homogen(x-2,y+2))< mnh) {
                mnh = th;
                dir = 2;
        }
        if ((th=homogen(x-2,y-2))< mnh) {
                mnh = th;
                dir = 3;
        }
        if ((th=homogen(x+2,y-2))< mnh) {
                mnh = th;
                dir = 4;
        }
        return(dir);
}

unsigned int homogen(x,y)
unsigned int x,y;
{
    unsigned int xp = x+1,xm=x-1,yp=y+1,ym=y-1;
    return(abs( (int) (Ak[xp][yp]+Ak[xm][yp]-Ak[xp][ym]-Ak[xm][ym]) ) +
        abs( (int) (Ak[xp][yp]+Ak[xp][ym]-Ak[xm][yp]-Ak[xm][ym])));
}

void average_pic()
{
        unsigned int i,j;
        unsigned int te;
        for (i=0; i<Height;i++)
                for (j=0;j<Width;j++)
                        Ak[i][j] = (unsigned char) 0;
        for (i=1; i<Height-1; i++) {
                te = ipic[i-1][0]+ipic[i][0]+ipic[i+1][0] + ipic[i-1][1]+ipic[i][1]+ipic[i+1][1];
                te /= 6;
                Ak[i][0] = (unsigned char) te;
                te = ipic[i-1][Width-1]+ipic[i][Width-1]+ipic[i+1][Width-1];
                te += ipic[i-1][Width-2]+ipic[i][Width-2]+ipic[i+1][Width-2];
                te /= 6;
                Ak[i][Width-1] = (unsigned char) te;
                te = ipic[0][i-1]+ipic[0][i]+ipic[0][i+1];
                te = ipic[1][i-1]+ipic[1][i]+ipic[1][i+1];
                te /= 6;
                Ak[0][i] = (unsigned char) te;
                te = ipic[Height-1][i-1]+ipic[Height-1][i]+ipic[Height-1][i+1];
                te = ipic[Height-2][i-1]+ipic[Height-2][i]+ipic[Height-2][i+1];
                te /= 6;
                Ak[Height-1][i] = (unsigned char) te;
                for(j=1;j<Width-1; j++) {
                        te = ipic[i-1][j-1]+ipic[i-1][j]+ipic[i-1][j+1];
                        te += ipic[i][j-1]+ipic[i][j]+ipic[i][j+1];
                        te += ipic[i+1][j-1]+ipic[i+1][j]+ipic[i+1][j+1];
                        te /= 9;
                        Ak[i][j] = (unsigned char)  te;
                }
        }
        te = (ipic[0][0]+ipic[0][1]+ipic[1][0]+ipic[1][1])/4;
        Ak[0][0] = (unsigned char) te;
        te = (ipic[0][Width-1]+ipic[0][Width-2]+ipic[1][Width-1]+ipic[1][Width-2])/4;
        Ak[0][Width-1] = (unsigned char) te;
        te = (ipic[Height-1][0]+ipic[Height-1][1]+ipic[Height-2][0]+ipic[Height-2][1])/4;
        Ak[Height-1][0] = (unsigned char) te;
        te = (ipic[Height-1][Width-1]+ipic[Height-1][Width-2]+ipic[Height-2][Width-1]+ipic[Height-
2][Width-2])/4;
        Ak[Height-1][Width-1] = (unsigned char) te;
}

#endif
```

APPENDIX C

Source Code Listing for Device Drivers

```
# ----- MAKE.MC -----

# Makefile for BP
#
CC      = cl
#CFLAGS= /AL /FPi87 /G2 /Zi /Od /W3
CFLAGS= /AL /FPi87 /G2 /Oailt /Gs /W3

.c.obj:
        $(CC) /c $(CFLAGS) $*.c

.obj.exe:
#        link /CO /NOD $*.obj util.obj,$*.exe,nul.map,llibc7p c:\os2\doscalls.lib,def.def
         link /NOD $*.obj util.obj,$*.exe,nul.map,llibc7p c:\os2\doscalls.lib,def.def

.c.exe:
        $(CC) /c $(CFLAGS) $*.c
#       link /CO /NOD $*.obj,$*.exe,nul.map,llibc7p c:\os2\doscalls.lib util.lib,def.def
        link /NOD $*.obj,$*.exe,nul.map,llibc7p c:\os2\doscalls.lib util.lib,def.def
        del $*.obj


load.exe: load.c

save.exe: save.c

enlarge.exe: enlarge.c

zoom.exe: zoom.c

equal.exe: equal.c

enhred.exe: enhred.c

clrbuf.exe: clrbuf.c

acquire.exe: acquire.c

on.exe: on.c

off.exe: off.c

paint.exe: paint.c

dual.exe: dual.c

half.exe: half.c
```

```
/* ----- File ACQUIRE.C ----- */

#include <stdio.h>
#include <conio.h>
#include "util.h"
void main(void);

void main()
{
int ha;
au_init(&ha);
au_output(ha,0x8188);
printf("Now in passthru mode.\n");
printf("Hit any key to freeze image.\n");
getch();
au_output(ha,0x8000);
au_end(ha);
}
```

```
/* ----- File CLRBUF.C ----- */

#include <stdio.h>
#include <stdlib.h>
#include "util.h"
void main(int,char *[]);

void main(argc,argv)
int argc;
char *argv[];
{
int ha;
int buff,i;
if (argc==1) exit(0);
au_init(&ha);
for (i=1;i<argc;i++) {
        buff=atoi(argv[i]);
        printf("Clearing buffer %d...\n",buff);
        au_buf_clear(ha,buff);
        }
au_end(ha);
}
```

```
/* ----- File DUAL.C ----- */

#include <stdio.h>
#include <stdlib.h>
#include "util.h"

typedef int WORD;
typedef long DWORD;

unsigned char huge pic[512][512];
void main(int,char *[]);

void main(argc,argv)
int argc;
char *argv[];
{
FILE *fp1,*fp2;
int i,j,ha;
if (argc!=3) exit(0);
au_init(&ha);
printf("Loading %s...\n",argv[1]);
printf("Loading %s...\n",argv[2]);
fp1=fopen(argv[1],"rb");
fp2=fopen(argv[2],"rb");
fseek(fp1,512L,SEEK_SET);
fseek(fp2,512L,SEEK_SET);
for (i=0;i<512;i++)
        for (j=0;j<256;j++) {
                pic[i][j] = (unsigned char) fgetc(fp1);
                fgetc(fp1);
                pic[i][256+j]=(unsigned char) fgetc(fp2);
                fgetc(fp2);
                }
au_writescr(ha,pic,0);
for (i=0;i<512;i++)
        for (j=0;j<256;j++) {
                pic[i][j] = (unsigned char) fgetc(fp1);
                fgetc(fp1);
                pic[i][256+j]=(unsigned char) fgetc(fp2);
                fgetc(fp2);
                }
au_writescr(ha,pic,1);
for (i=0;i<512;i++)
        for (j=0;j<256;j++) {
                pic[i][j] = (unsigned char) fgetc(fp1);
                fgetc(fp1);
                pic[i][256+j]=(unsigned char) fgetc(fp2);
                fgetc(fp2);
                }
au_writescr(ha,pic,2);
fclose(fp1);
fclose(fp2);
au_end(ha);
}
```

```
/* ----- File ENHRED.C ----- */

#include <stdio.h>
#include "util.h"

#define LORED 40
#define HIRED 185


int status, colorval;
int values[256], actrgn[4];
char filename[20], comment[100], ch;
unsigned char huge huepic[512][512];
unsigned char huge satpic[512][512];
void main(void);

void main()
{
int i,j,ha;
au_init(&ha);
au_buf_clear(ha,3);
au_readscr(ha,huepic,2);
au_readscr(ha,satpic,1);
printf("Enhancing red areas of image...\n");
for (i=0;i<480;i++)
        for (j=0;j<512;j++) {
                if ((huepic[i][j] <=LORED) || (huepic[i][j]>=HIRED)) {
                        huepic[i][j]=255;
                        }
                else
                        satpic[i][j]=0;
                }
au_writescr(ha,satpic,1);
au_writescr(ha,huepic,2);
printf("Bright red areas show evidence of liquid penetrant.\n");
au_end(ha);
}
```

```
/* ----- File ENLARGE.C ----- */

#include <stdio.h>
#include <conio.h>
#include "util.h"

unsigned char huge ipic[512][512];
unsigned char huge opic[512][512];
void main(void);
void enlarge(int buf);

int ha;

void main()
{
int i;
        /* Initialization */

au_init(&ha);
for (i=0;i<3;i++) {
        printf("Enlarging buffer %d...\n",i);
        enlarge(i);
        }
au_end(ha);

}

void enlarge(buf)
int buf;
{
int i,j;
unsigned char ch;
au_readscr(ha,ipic,buf);
for (i=0;i<512;i+=2)
        for (j=0;j<512;j+=2) {
                ch = ipic[128+i/2][128+j/2];
                opic[i][j] = ch;
                opic[i][j+1] = ch;
                opic[i+1][j] = ch;
                opic[i+1][j+1] = ch;
                }
au_writescr(ha,opic,buf);
}
```

```
/* ----- File EQUAL.C ----- */

#include <stdio.h>
#include <stdlib.h>
#include "util.h"

unsigned char huge ipic[512][512];
void main(void);

void main() {
int ha;
unsigned int bins=256;
unsigned int hist[256],j,map[256],binpos,i;
unsigned long binwidth,sum,maxbinval,li,lj;
au_init(&ha);
au_readscr(ha,ipic,0);
printf("Initializing...\n");
for (i=0;i<256;i++)
        hist[i]=0;
maxbinval=binwidth=(512L*512L)/(unsigned long) bins;
binpos=0;
sum=0L;
printf("Entering histogram...\n");
for (li=0L;li<512L;li++) {
        for (lj=0L;lj<512L;lj++) {
                i=(unsigned int)ipic[li][lj];
                ++hist[i];
                }
        }
printf("Entering mapping...\n");
for (j=0;j<256;j++) {
        loop:
        if ((sum+(unsigned long)hist[j]) <= maxbinval) {
                map[j]=binpos*256/bins;
                sum += (unsigned long) hist[j];
                }
        else {
                binpos++;
                maxbinval += binwidth;
                goto loop;
                }
        }
printf("Entering remapping...\n");
for (li=0L;li<512L;li++) {
        for (lj=0L;lj<512L;lj++) {
                i = (unsigned int) ipic[li][lj];
                ipic[li][lj] = (char) map[i];
                }
        }
au_writescr(ha,ipic,0);
au_end(ha);
}
```

```
/* ----- File HALF.C ----- */

#include <stdio.h>
#include <stdlib.h>
#include "util.h"

typedef int WORD;
typedef long DWORD;

unsigned char huge pic[512][512];
void main(int,char *[]);

void main(argc,argv)
int argc;
char *argv[];
{
FILE *fp1,*fp2;
int i,j,ha;
if (argc!=3) exit(0);
au_init(&ha);
printf("Loading %s...\n",argv[1]);
printf("Loading %s...\n",argv[2]);
fp1=fopen(argv[1],"rb");
fp2=fopen(argv[2],"rb");
fseek(fp1,512L,SEEK_SET);
fseek(fp2,512L,SEEK_SET);
for (i=128;i<384;i++) {
        for (j=0;j<256;j++) {
                pic[i][j] = (unsigned char) fgetc(fp1);
                fgetc(fp1);
                pic[i][256+j]=(unsigned char) fgetc(fp2);
                fgetc(fp2);
                }
        fseek(fp1,512L,SEEK_CUR);
        fseek(fp2,512L,SEEK_CUR);
        }
au_writescr(ha,pic,0);
for (i=128;i<384;i++) {
        for (j=0;j<256;j++) {
                pic[i][j] = (unsigned char) fgetc(fp1);
                fgetc(fp1);
                pic[i][256+j]=(unsigned char) fgetc(fp2);
                fgetc(fp2);
                }
        fseek(fp1,512L,SEEK_CUR);
        fseek(fp2,512L,SEEK_CUR);
        }
au_writescr(ha,pic,1);
for (i=128;i<384;i++) {
        for (j=0;j<256;j++) {
                pic[i][j] = (unsigned char) fgetc(fp1);
                fgetc(fp1);
                pic[i][256+j]=(unsigned char) fgetc(fp2);
                fgetc(fp2);
                }
        fseek(fp1,512L,SEEK_CUR);
        fseek(fp2,512L,SEEK_CUR);
        }
au_writescr(ha,pic,2);
fclose(fp1);
fclose(fp2);
au_end(ha);
}
```

```
/* ----- File LOAD.C ----- */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "util.h"

typedef int WORD;
typedef long DWORD;
void main(int argc, char *argv[]);

void main(argc,argv)
int argc;
char *argv[];
{
char fname[130];
int ha;
if (argc==2) strcpy(fname,argv[1]);
else {
        printf("Enter file to load: ");
        scanf("%s",fname);
        }
au_init(&ha);
printf("Loading %s...\n",fname);
au_load(ha,fname);
au_end(ha);
}
```

```
/* ----- File OFF.C ----- */

#include "util.h"
void main(void);

void main()
{
int ha;
au_init(&ha);
au_output(ha,0x0000);
au_end(ha);
}
```

```
/* ----- File ON.C ----- */

#include "util.h"
void main(void);

void main()
{
int ha;
au_init(&ha);
au_output(ha,0x8000);
au_end(ha);
}
```

```
/* ----- File PAINT.C ----- */

#define INCL_BASE
#define INCL_DOSDEVICES
#include <stdio.h>
#include <stdlib.h>
#include <os2.h>
#include "util.h"

typedef int WORD;
typedef long DWORD;

unsigned char huge pic[512][512];

MOUEVENTINFO MouEvent;

main()
{
int i,j;
char OutStr[40];
USHORT Cell=0x0720;
HMOU MouHandle;
USHORT status;
SHORT WaitOption=1;
int plan[4];
au_init();
plan[0]=plan[1]=plan[2]=plan[3]=0;
status=MouOpen(0L,&MouHandle);
au_set_ovl_plns(plan);
au_set_ovl_clrs(plan);
au_set_const(3,255);
getch();
if(status) {
        printf("MouOpen Failed.\n");
        exit(1);
        }

VioScrollUp(0,0,-1,-1,-1,&(char)Cell,0);

puts("Press Both Mouse Buttons To Exit");

MouDrawPtr(MouHandle);

do {
        sprintf(OutStr,"X=%2d Y=%2d", MouEvent.col,MouEvent.row);
        VioWrtCharStr(OutStr,strlen(OutStr),0,0,0);
        MouReadEventQue(&MouEvent,&WaitOption,MouHandle);
        } while((MouEvent.fs & 0x14) != 0x14);
MouClose(MouHandle);
puts("Have a Mice Day!");
au_end();
}
```

```
/* ----- File SAVE.C ----- */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "util.h"

typedef int WORD;
typedef long DWORD;
void main(int argc,char *argv[]);

void main(argc,argv)
int argc;
char *argv[];
{
char fname[130];
int ha;
if (argc==2) strcpy(fname,argv[1]);
else {
        printf("Enter file to save: ");
        scanf("%s",fname);
        }
au_init(&ha);
printf("Saving %s...\n",fname);
au_save(ha,fname);
au_output(ha,0x8000);
au_end(ha);
}
```

```
/* ----- File ZOOM.C ----- */

#include <stdio.h>
#include <conio.h>
#include "util.h"

unsigned char huge ipic[512][512];
unsigned char huge opic[512][512];
void main(void);
void zoom(int);

int ha;

void main()
{
int i;
        /* Initialization */
au_init(&ha);
for (i=0;i<3;i++) {
        printf("Zooming buffer %d...\n",i);
        zoom(i);
        }
au_end(ha);
}

void zoom(buff)
int buff;
{
int i,j,x,y;
au_readscr(ha,ipic,buff);
for (i=0;i<512;i+=2)
        for (j=0;j<512;j+=2) {
                x=128+i/2;
                y=128+j/2;
                opic[i][j] = ipic[x][y];
                opic[i][j+1]= (ipic[x][y]+ipic[x][y+1])/2;
                opic[i+1][j]= (ipic[x][y]+ipic[x+1][y])/2;
                opic[i+1][j+1]= (ipic[x][y]+ipic[x+1][y+1])/2;
                }
au_writescr(ha,opic,buff);
}
```

Final Report Distribution:

AP29F:	1 copy
CN22D:	5 copies
AT01:	1 copy
CC01:	1 copy
EM12-26, Womac:	1 copy
EB44, Boglio:	6 copies
NTIS:	2 copies (one unbound)
DCASMA, Birmingham:	1 copy
Remaining Fee:	$ 1063.95

Crown Undercut Edge Crack